LENUŢA ALBOAIE, SABIN BURAGA

Servicii Web

Seria *Web* este coordonată de dr. Sabin Buraga
(Facultatea de Informatică, Universitatea „Al.I. Cuza", Iași)

Lenuța Alboaie este doctorandă la Universitatea „Al.I. Cuza" din Iași și are ca domenii de interes serviciile Web și sistemele distribuite. În prezent, este cercetător programator la Axiologic Research, SRL și cadru asociat al Facultății de Informatică a Universității „Al.I. Cuza".

Sabin Buraga este lector doctor la Facultatea de Informatică a Universității „Al.I. Cuza" din Iași, fiind titularul cursurilor „Tehnologii Web", „Semantic Web", „Interacțiune om-calculator" și „Rețele de calculatoare". Este laureat al Premiului „Gheorghe Cârtianu" al Academiei Române (2005). Mai multe amănunte privitoare la activitățile sale sunt disponibile pe Web la adresa *http://www.infoiasi.ro/~busaco*. La Editura Polirom a mai publicat: *Atelier de programare în rețele de calculatoare* (în colaborare, 2001), *Programare Web în bash și Perl* (în colaborare, 2002), *Proiectarea siturilor Web. Design și funcționalitate* + CD (2002), *Aplicații Web la cheie* (coord., 2003), *Utilizare Linux* (în colaborare, 2004), *Situri Web la cheie. Soluții profesionale de implementare* (coord., 2004), *Proiectarea siturilor Web. Design și funcționalitate* + CD (ediția a II-a, 2005) și *Primii pași în Linux* (în colaborare, 2006).

© 2006 by Editura POLIROM

www.polirom.ro

Editura POLIROM
Iași, B-dul Carol I nr. 4, P.O. BOX 266, 700506
București, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33, O.P. 37; P.O. BOX 1-728, 030174

Descrierea CIP a Bibliotecii Naționale a României:

ALBOAIE, LENUȚA

Servicii web: concepte de bază și implementări / Lenuța Alboaie, Sabin Buraga. Iași: Polirom, 2006
ISBN (10) 973-681-522-6
ISBN (13) 978-973-681-522-5

I. Buraga, Sabin

004.42
004.738.5

Printed in ROMANIA

LENUȚA ALBOAIE • SABIN BURAGA

SERVICII WEB

Concepte de bază și implementări

POLIROM
2006

Cuprins

Prefață 9

Capitolul 1. **Prezentare generală a serviciilor Web** 13
1. Preambul 13
1.1. Protocolul HTTP pe scurt 13
2. Arhitectura orientată spre servicii Web 19
2.1. Introducere 19
2.2. Programarea aplicațiilor Web 20
2.3. Servicii Web. Punerea problemei 21
2.4. Conceptul de serviciu Web bazat pe XML 26
2.5. Modalitățile de implementare și exploatare 34
Referințe 36

Capitolul 2. **Familia XML** 39
1. Preliminarii 39
2. XML (*Extensible Markup Language*) 40
2.1. Caracterizare și trăsături esențiale 40
2.2. Părți componente ale unui document XML 41
2.3. Membrii constituenți ai familiei XML 43
2.4. Spațiile de nume XML 44
2.5. XML Infoset 45
2.6. Validarea documentelor XML 47
2.7. Procesarea documentelor XML 54
Referințe 59

Capitolul 3. **Descrierea serviciilor Web** 61
1. Punerea problemei 61
2. WSDL (*Web Service Description Language*) 62
2.1. Modelul conceptual al WSDL 64
2.2. Diferențele dintre specificațiile WSDL 1.1 și WSDL 2.0 72
2.3 Exemple de documente WSDL 72
2.4. Tipuri de operații și mesaje 76
2.5 Platforme și instrumente de lucru cu documentele WSDL 78
Referințe 84

Capitolul 4. **Protocolul SOAP** 87
1. Evoluția protocoalelor bazate pe XML 87
1.1. Prezentare succintă a protocoalelor din prima generație 87
1.2. Dezavantajele protocoalelor din prima generație 90
2. SOAP – scurtă istorie 91
3. SOAP – o vedere de ansamblu 92
4. Forma generală a unui mesaj SOAP. Procesarea datelor de către serverele SOAP 94
5. Modelul SOAP – mesaje și codificări 98
5.1. Mesajele SOAP 98
5.2. Elementul *Envelope* 99
5.3. Elementul *Header* 100
5.4. Corpul mesajului SOAP. Elementul *Body* 104
5.5. Maniera de codificare a mesajelor – SOAP *Encoding* 108
5.6. Reguli de serializare 116
6. Maniera de transport al mesajelor SOAP 119
6.1. Relația dintre SOAP și HTTP 119
6.2. Relația dintre SOAP și transportul de mesaje de poștă electronică (e-mail) 122
7. Apelul de proceduri la distanță via SOAP RPC 124
8. Versiuni SOAP 130
9. Intermediarii SOAP 130
10. O privire asupra SOAP 1.1. O comparație cu SOAP 1.2 133
Referințe 140

Capitolul 5. **Descoperirea serviciilor** 143
1. Descoperirea serviciilor Web prin UDDI 143
1.1. Scurt istoric 144
1.2. Concepte de bază ale UDDI 144
1.3. Structura *tModel* 150
1.4. Informații privitoare la publicarea serviciului 152
1.5. Taxonomia entităților UDDI 153
1.6. Interfețe de programare pentru UDDI 155
1.7 UDDI și SOAP 158
1.8. UDDI și WSDL 159
1.9. Crearea propriului registru UDDI folosind *jUDDI* 171
2. WSIL (*Web Service Inspection Language*) 179
Referințe 182

Capitolul 6. **Dezvoltarea și utilizarea serviciilor Web** 185
1. Dezvoltarea și invocarea de servicii Web folosind instrumente *open source* 185
1.1. Introducere 185
1.2. Furnizarea stocurilor de portocale via un serviciu Web scris în PHP 5 186

1.3. Un client PHP pentru serviciul Web privitor la portocale 188
1.4. Un serviciu de manipulare a fișierelor implementat în Perl cu *SOAP::Lite* 189
1.5 Scrierea în Perl a clientului SOAP 192
1.6. Invocarea serviciului de către un client PHP 196
2. Inter-operabilitatea între diverse tehnologii, instrumente și limbaje de programare a serviciilor Web 199
2.1. Dezvoltarea de servicii Web în *.NET Framework* 199
2.2. Un client C# pentru serviciul creat 210
2.3. Accesarea dintr-un *script* Perl a serviciului Web implementat în .NET 212
2.4. Invocarea din PHP a serviciului Web 213
2.5. Invocarea de servicii Web externe 216
2.6. Folosirea instrumentului *gSOAP* pentru C/C++ 224
2.7. Implementarea serviciilor Web în Java 228
2.8. Suportul oferit de mediul Delphi pentru crearea și utilizarea serviciilor Web 237
3. Invocarea serviciilor Web în contextul AJAX 245
3.1. Punerea problemei 245
3.2. Caracteristici importante ale suitei de tehnologii AJAX 245
3.3. Invocarea asincronă a unui serviciu Web via AJAX 249
3.4. Utilizarea bibliotecii *Prototype* 255
3.5. Invocarea directă, din JavaScript, a unui serviciu Web 259
3.6. Transferuri asincrone prin *Atlas ASP.NET* 263
Referințe 271

Capitolul 7. **Retrospectivă și perspective** 273
1. Procesul de dezvoltare a serviciilor Web 273
2. Alte inițiative vizând serviciile Web 275
2.1. Privire de ansamblu 275
2.2. Adresarea serviciilor prin *WS-Addressing* 276
2.3. Descoperirea serviciilor 276
2.4. Coordonarea serviciilor Web 278
2.5. Accesarea resurselor serviciilor Web 283
2.6. Accesarea meta-datelor asociate serviciilor 285
2.7. Securitatea serviciilor Web 286
2.8. Asigurarea inter-operabilității 295
2.9 Serviciile Web în contextul proceselor de afaceri 298
2.10. Servicii Web pentru *grid* 302
3. SOA în contextul noului Web 304
Referințe 307

Acronime 311

Bibliografie generală 317

Prefață

Misiunea cărții de față este, cel puțin pentru autori, atât una relativ dificilă, cât – mai ales! – atrăgătoare: ne propunem să prezentăm sistematic unul dintre domeniile importante, dinamice și de succes ale tehnologiilor actuale – *serviciile Web*. Astfel, adresăm acest volum viitorului sau actualului programator Web care dorește să se inițieze în tehnologiile și metodologiile de dezvoltare a serviciilor Web bazate preponderent pe SOAP. Desigur, nu uităm nici abordarea REST, concretizată în ilustrarea invocării de servicii în mod asincron via suita de tehnologii AJAX.

Plecând de la metafora că putem asemăna un serviciu Web cu o portocală, vom încerca să „decojim" straturile – poate uneori mai aspre – ale specificațiilor privitoare la descrierea prin WSDL a interfeței și implementării serviciului, la protocolul de transport SOAP și la descoperirea prin diverse metode (*e.g.*, UDDI) a serviciilor disponibile în regim public sau privat. Într-un anumit moment vom ajunge și la miez, adică tocmai la prezentarea limbajelor, instrumentelor și platformelor ce oferă suport pentru crearea și invocarea de servicii Web.

Programatorii se vor putea delecta cu diverse biblioteci, *framework*-uri și servere de aplicații, prinzând gustul implementării de servicii (cu mult) mai complexe. Nu uităm să enumerăm noțiunile de bază privitoare la familia XML – baza pe care se fundamentează întreg eșafodajul de limbaje și inițiative privitoare la serviciile Web – ori să „tragem cu ochiul" la livada de portocali actuali, (re)prezentând perspectivele domeniului.

Materialul îi are drept destinatari pe toți cei interesați de tehnologiile Web în general și de servicii Web în special: studenți de la facultăți de profil, elevi din clasele terminale, dezvoltatori din cadrul companiilor, alte categorii de specialiști în informatică sau în domenii conexe. Notăm, *en passant*, că una dintre cerințele obligatorii ale secțiunii *Software Design* a deja bine-cunoscutei competiții internaționale *Imagine Cup* este ca fiecare echipă concurentă să implementeze cel puțin un serviciu Web…

Vom descrie în continuare structura lucrării de față.

Capitolul 1 conține o prezentare generală a serviciilor Web, în contextul dezvoltării aplicațiilor distribuite bazate pe XML. Tot aici realizăm o trecere în revistă a caracteristicilor principale ale protocolului HTTP, introducem SOA (arhitectura orientată spre servicii) și discutăm necesitatea existenței tehnologiilor SOAP, WSDL, UDDI, REST.

Al doilea capitol se focalizează asupra familiei de limbaje XML. Descriem sintaxa, spațiile de nume, manierele de validare și tehnicile de procesare a documentelor XML. Insistăm asupra unor concepte privitoare la XML Schema și modelul DOM, deoarece ne vor fi de folos pe parcursul cărții.

Cel de-al treilea capitol este dedicat manierei de descriere a serviciilor Web. În primul rând, ne oprim asupra limbajului WSDL: componente, tipuri de documente, de operații și de mesaje, exemple și instrumente de lucru.

În cadrul capitolului patru detaliem „inima" serviciilor Web – SOAP. După o succintă prezentare a protocoalelor de transport bazate pe XML, realizăm o privire de ansamblu a protocolului SOAP. De asemenea, printre altele, sunt descrise: structura și maniera de codificare a mesajelor, modul de procesare și transport ale datelor, relația dintre SOAP și HTTP și diferențele dintre versiunile în vigoare ale acestui important protocol.

Capitolul cinci are drept subiect descoperirea serviciilor Web. Prima parte se focalizează asupra UDDI, unul dintre cele mai populare mecanisme de căutare a serviciilor, mai ales prin prisma proceselor de afaceri pe care le modelează. După detalierea arhitecturii UDDI, a tipurilor de informații stocate și a intimităților privind funcționarea, oferim și o soluție practică de constituire a unui registru UDDI propriu prin intermediul instrumentului *open source jUDDI*. A doua parte a capitolului vizează descoperirea dinamică a serviciilor prin *WS-Inspection*.

Pentru unii dintre cititorii mai grăbiți, capitolul șase ar putea reprezenta cea mai incitantă parte a volumului, deoarece cuprinde „rețete" de „sădire" a serviciilor Web – fie ele referitoare la portocale (albastre) sau la alte aspecte – și de „consumare" a funcționalităților oferite de acestea. Am folosit majoritatea limbajelor de programare (C/C++, C#, Java, Perl, PHP, Visual Basic), a instrumentelor (*Apache Axis2*, *gSOAP*, *NuSOAP*, *SOAP::Lite*) și mediilor de dezvoltare (.NET, Java, Delphi) actuale, fie ele comerciale sau liber disponibile. Tot aici punctăm etapele care trebuie parcurse pentru accesarea serviciilor Web externe (Google și XMethods) și ilustrăm principiile de bază ale suitei de tehnologii AJAX. Prezentăm, de asemenea, maniera de invocare asincronă a serviciilor Web, fie direct – via programe JavaScript –, fie pe baza unor biblioteci și *framework*-uri precum *Prototype* și *Atlas ASP.NET*.

Ultimul capitol realizează o recapitulare a tematicilor atinse și trasează diverse direcții de evoluție, invitându-i pe cei interesați să aprofundeze domeniul. Astfel, prezentăm succint inițiative industriale ca *WS-Addressing*, *WS-Discovery*, *WS-Coordination* sau *WS-AtomicTransaction*. Suplimentar, punem în discuție aspecte referitoare la ingineria, securitatea și asigurarea inter-operabilității serviciilor Web. Spre final, „degustăm" rapid rolul serviciilor Web în cadrul proceselor de afaceri, al sistemelor de tip *grid* și al noului Web (ale cărui tendințe sunt cunoscute sub denumirea generică de Web 2.0, etapă preliminară Web-ului semantic).

Volumul se încheie cu un set cuprinzător de acronime folosite în text și, în mod firesc, cu o bibliografie generală.

Această carte nu ar fi ajuns la forma actuală fără ajutorul oferit – atât prin parcurgerea versiunilor preliminare, cât și prin formularea unor sugestii utile – de *Sînică Alboaie*, *Mihaela Brut*, *Diana Gorea*, *Adrian Iftene*, *Loredana* și *Ștefan Tanasă* și, nu în ultimul rând, *Cosmin Vârlan*. Suntem recunoscători profesorilor noștri *Toader Jucan*, *Cornelius Croitoru*, *Dorel Lucanu* și *Henri Luchian*, de la Facultatea de Informatică a Universității „Alexandru Ioan Cuza" din Iași. De asemenea, mulțumim familiilor noastre și echipei de profesioniști a Editurii Polirom.

La finalul acestei prefețe trebuie să menționăm contribuțiile autorilor, după cum urmează: capitolele 3, 4 și 5 au fost redactate în principal de *Lenuța Alboaie*, care a realizat și implementarea exemplelor de servicii Web scrise în limbajele C/C++, Java și Visual Basic .NET din cadrul capitolului 6. Restul materialului, inclusiv ilustrațiile, a fost preponderent conceput de *Sabin Buraga*.

Pentru experimentarea codului-sursă al programelor incluse, nu ezitați să vizitați adresa *http://www.infoiasi.ro/~busaco/books/ws/*. De asemenea, vă invităm să ne contactați prin poșta electronică la *adria@infoiasi.ro* și, respectiv, *busaco@infoiasi.ro*.

Autorii
Iași, 3 septembrie 2006

Capitolul 1

Prezentare generală a serviciilor Web

„Cea mai bună cale de a prezice viitorul este să-l inventăm."

(Alan Key)

1. Preambul

Spațiul World Wide Web reprezintă un *sistem de distribuție locală sau globală a informațiilor hipermedia*, văzut ca univers informațional compus din elemente de interes, denumite *resurse*, desemnate de identificatori globali – URI (*Uniform Resource Identifiers*).

Din punct de vedere tehnic, Web-ul pune la dispoziție un sistem global și standardizat de comunicare multimedia (*summum* de media), informațiile fiind organizate asociativ și distribuite în funcție de cererile utilizatorilor, funcționând conform modelului client/server. Prin definiție, *clientul* (în cazul nostru denumit și navigator sau agent-utilizator Web) solicită servicii (informații) de la componenta *server*. Serverul răspunde așadar cererilor clienților, protocolul folosit fiind, uzual, HTTP (*HyperText Transfer Protocol*).

1.1. *Protocolul HTTP pe scurt*

Dată fiind importanța protocolului HTTP, vom realiza în continuare o trecere în revistă a caracteristicilor sale importante.

Protocolul HTTP, standardizat de documentul RFC 2616, este folosit în special pentru hipertext, dar poate fi considerat drept protocol generic, baza comunicării într-un sistem distribuit de management al datelor, în cazul WWW fiind vorba în principal de facilitarea transferului de date între clienții și serverele Web. Fiind un protocol utilizat în Internet, HTTP se situează la nivelul de aplicații al stivei de protocoale TCP/IP (*Transmission Control Protocol/Internet Protocol*), putând fi considerat fiabil (*reliable*).

Concepte fundamentale

Conceptele de bază sunt *cererea* și *răspunsul*: un *client* Web trimite un mesaj (cererea) la un *server*. Mesajul conține identificatorul resursei dorite, dat sub forma unui URI (*Uniform Resource Identifier*), metoda de acces folosită, versiunea protocolului, precum și o serie de meta-informații care pot fi utile serverului. Răspunsul serverului cuprinde un cod indicând starea serverului după interpretarea cererii, un mesaj explicativ pentru codul de stare transmis, meta-informațiile care vor fi procesate de către client și, eventual, un conținut (*e.g.*, resursa solicitată).

În general, o sesiune de comunicare HTTP este inițiată de către client și constă din solicitarea unei resurse (o pagină Web, uzual) identificată unic pe un server cunoscut. Acesta este numit și *server de origine* datorită faptului că în comunicarea între client și server pot să apară unul sau mai mulți intermediari: *proxy* (numit și *server proxy*), *poartă* (*gateway*) sau *tunel* (*tunnel*). Entitatea *proxy* reprezintă un intermediar care retrimite un mesaj HTTP, eventual modificând o parte a sa. *Poarta* semnifică un intermediar care se poate situa înaintea unui server de origine și să se identifice drept acesta, clientul Web necunoscând acest aspect. *Tunelul* este un intermediar care nu schimbă conținutul mesajului, ci are rolul exclusiv de retransmitere a lui; de exemplu, tunelul poate fi folosit pentru (de)criptarea mesajelor vehiculate între server și client în cadrul unui intranet/extranet.

Un alt concept important este cel de *cache*, desemnând un depozit local de stocare (în memorie, pe disc) a mesajelor (datelor) la nivel de server/client. Un *proxy* deține obligatoriu un *cache*, denumit și *sistem de cache*.

Un exemplu tipic de *cerere-răspuns* este următorul, în care cererea – formulată de un browser – are forma:

```
GET /catalog_produse.html HTTP/1.1
Host: www.portocale.info
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.0.5) Gecko/20060719 Firefox/1.5.0.5
Accept: text/html, image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate, compress, identity
Connection: Keep-Alive
```

Un posibil răspuns – obținut din partea serverului Web – ar putea fi următorul (am omis conținutul propriu-zis al documentului solicitat):

```
Date: Tue, 22 Aug 2006 07:17:13 GMT
Server: Apache/2.0.54 (Win32) PHP/5.0.4
```

```
Accept-Ranges: bytes
Content-Length: 201
Keep-Alive: timeout=15, max=74
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
...
```

Adresarea resurselor via URI

Modalitatea de adresare a resurselor Web este reprezentată de *identificatorii uniformi de resurse* – URI (*Uniform Resource Identifiers*). Mulțimea URI e compusă din localizatori uniformi de resurse – URL (*Uniform Resource Locator*) – și nume uniforme de resurse – URN (*Uniform Resource Name*).

Adresele de tip URL sunt folosite în localizarea unei resurse în rețea (în special în Internet) prin protocolul HTTP, având forma bine-cunoscută *http://server:port/cale?interogare*, unde în mod implicit, dacă nu e precizat, portul se consideră 80, iar *cale* reprezintă un șir de directoare delimitate de „/”, terminat eventual cu numele fișierului care stochează resursa adresată prin intermediul URL-ului. Componenta *interogare* este opțională și desemnează un șir suplimentar, incluzând informații interpretate de resursă (de cele mai multe ori de un *script*).

De precizat faptul că o serie de caractere sunt rezervate, cum ar fi simbolurile „;”, „/”, „?”, „:”, „@”, „&”, „=”, „+” și „$”. Aceste caractere speciale sunt codificate conform unei metode denumite *URL encoding* în care caracterul în cauză este substituit de codul său numeric, în baza 16, prefixat de semnul „%” (de exemplu, „~” devine „%7E”).

Un URL poate avea drept sufix un identificator de fragment, precedat de caracterul „#”, cu scopul de a face referire directă la o parte constituentă a resursei adresate de acel URL. Drept exemplificare, se poate da URL-ul *http://www.portocale.info/catalog_produse.html#albastre*.

O adresă de tip URN desemnează un subset al URI care rămâne permanent și unic, chiar dacă resursa a dispărut ori a devenit inaccesibilă. URN-ul se utilizează mai ales pentru a desemna entități (tipuri de date, spații de nume etc.) folosite de anumite aplicații Web. Drept exemplificări, enumerăm:

– *urn:mozilla:package:communicator* identifică pachetele software ale suitei Mozilla;
– *urn:schemas-microsoft-com:datatypes* desemnează tipurile de date definite de Microsoft;
– *urn:ISBN:978-973-46-0249-0* desemnează numărul ISBN (*International Standard Book Number*) asociat unei cărți.

Mai general, adresele URI pot preciza resurse care pot fi accesate prin intermediul unor diverse protocoale (uzual, cele bazate pe TCP/IP). Astfel, forma generică

a unui identificator uniform de resurse este *schema://autoritate/cale?interogare*. O serie de exemple de scheme sunt:

– *file:///tmp/* – schemă ce desemnează resurse de tip fișier din cadrul sistemului de stocare de la nivelul clientului (aici, directorul */tmp*);
– *ftp://ftp.funet.fi/pub/README.txt* – schemă utilizată pentru serviciile protocolului de transfer de fișiere (FTP);
– *https://www.infoiasi.ro/~busaco/teach/* – schemă pentru serviciile protocolului securizat HTTPS – HTTP folosit în conjuncție cu SSL (*Secure Sockets Layer*) sau TLS (*Transport Layer Security*);
– *mailto:busaco@infoiasi.ro* – schemă *mailto* pentru poșta electronică;
– *tag:blogger.com,2006:blog-333* – schemă destinată să identifice în mod unic (spațial și temporal) o anumită resursă, într-o manieră convenabilă pentru utilizator (în acest caz, este vorba de subiectele de interes ale unui *blog*);
– *uuid:d52c-e89c-01f8-3b53-a25e-89cf-a5bb-ad17* – schemă care specifică un identificator unic universal (UUID – *Universal Unique IDentifier*) ce desemnează o anumită resursă; acest identificator se reprezintă prin 32 de cifre în baza 16 și se regăsește uneori și sub denumirea de GUID (*Globally Unique IDentifier*).

Maniera de codificare a conținutului

Pentru ca datele să fie corect interpretate de toate entitățile participante la conversația prin rețea, indiferent de platforma hardware și software, ele trebuie să respecte aceeași codificare.

Astfel, se definește conceptul de *set de caractere*, pentru a putea interpreta corect datele schimbate via protocolul HTTP între două platforme diferite (*e.g.*, un server rulând pe un sistem Linux cu un navigator Web de pe un dispozitiv mobil), având reprezentări diferite ale datelor. Informația e transmisă sub forma unui șir de caractere urmând ca entitatea de la celălalt capăt, folosind indicațiile despre setul de caractere, să realizeze transformarea din șir de octeți în șir de caractere. Setul de caractere implicit este ISO-8859-1, dar există o multitudine de alte seturi (de exemplu, ISO-8859-2 denumit și Latin-2, care oferă printre altele și reprezentarea diacriticelor din limba română).

Protocolul HTTP respectă seturile de caractere definite de specificațiile MIME (*Multipurpose Internet Mail Extensions*) descrise în documentele RFC 2045 și 2046. Conform standardului MIME, pentru fiecare resursă în parte se specifică tipul și subtipul acesteia. De exemplu, *text/html* pentru un document HTML, *text/plain* în cazul unui document text neformatat, *image/png* pentru o imagine în format PNG, *application/json* în situația datelor vehiculate via JSON (*JavaScript Object Notation*), *application/soap+xml* pentru mesajele SOAP (*Simple Object Access Protocol*) și multe altele.

De asemenea, mesajele pot fi codificate prin diverse metode, precum *gzip* ori *compress*, în vederea comprimării sau asigurării identității și/sau integrității.

Mesajele HTTP

Cererile și răspunsurile HTTP sunt vehiculate prin intermediul mesajelor. Astfel, mesajele HTTP sunt considerate de două tipuri: *cerere* provenită de la un client către un server și *răspuns* al serverului, trimis la acel client. Un mesaj e compus dintr-o succesiune de linii de text, delimitate de caracterele CRLF (*Carriage Return* și *Line Feed*). Prima linie semnifică o cerere efectuată de un client sau un cod de stare obținut de la server, urmată de un număr de atribute de antet.

Un antet (*header*) conține mai multe atribute care sunt folosite la completarea unei cereri sau a unui răspuns cu meta-informația necesară interpretării corecte a mesajului prin stabilirea unor valori specificate de către protocolul HTTP sau a unor protocoale definite de utilizator (de exemplu, SOAP). Un atribut este furnizat printr-un nume urmat de „:" și de o valoare (opțională, în unele cazuri).

O cerere e specificată de o metodă de acces, printre cele mai folosite fiind:

– *GET* reprezintă o cerere de accesare a unor informații (reprezentări de resurse). Un client HTTP (navigator, robot, program de descărcare, agregator de știri, *player* multimedia etc.) folosește metoda GET pentru a obține o anumită resursă, fie că ea reprezintă un fișier (document text, HTML, imagine PNG sau JPEG, aplicație, arhivă, document XML etc.), fie că indică execuția pe serverul Web a unui proces care va produce datele dorite (*e.g.*, invocarea unui *script* CGI sau a unui program PHP);
– *HEAD* este similară cu metoda GET, dar serverul va întoarce un mesaj având informații doar în antet. Meta-datele din anteturile HTTP din răspunsul la o cerere HEAD vor fi identice cu cele din răspunsul la o cerere GET. Utilitatea acestei metode constă în obținerea meta-datelor asociate unei resurse Web fără a transfera efectiv întreaga entitate, în vederea – de exemplu – a testării existenței resursei, a obținerii datei ultimei modificări sau furnizarea drepturilor de acces;
– *POST* este utilizată pentru a identifica dacă serverul acceptă o entitate înglobată în cerere. POST este proiectată să implementeze o metodă uniformă pentru funcții precum adnotarea resurselor, trimiterea datelor unui formular Web către server, extinderea unei baze de date printr-o operațiune de inserare, invocarea unui serviciu etc.

În cadrul unei cereri pot fi specificate diverse atribute, utilizate pentru a transmite serverului informații suplimentare privitoare la acea cerere și la client.

Se poate face o analogie între trimiterea unei metode HTTP și apelul unei funcții dintr-un limbaj de programare, unde atributele reprezintă parametrii de intrare.

După primirea și interpretarea unui mesaj de tip cerere și interpretarea lui, un server HTTP întoarce un mesaj denumit *răspuns*. Prima linie conține versiunea protocolului HTTP implementat de către server și continuă cu un cod de stare reprezentând un număr asociat de către specificația HTTP unei anumite situații a serverului în urma tratării unei cereri. Urmează un text explicativ pentru codul de stare, menit să clarifice situația exprimată de acesta.

Codul de stare este format din trei cifre organizate în categorii de stări; codurile din aceeași categorie se referă la stări similare. Ele se disting după prima cifră în modul următor:

– Coduri de informare (*1xx*) – furnizează informații privitoare la o anumită acțiune (cererea a fost primită, comunicația continuă). De exemplu: *100 Continue* (clientul poate continua cererea, trebuind să trimită următoarea parte a unui mesaj parțial).
– Coduri de succes (*2xx*) – raportează efectuarea cu succes a unei operațiuni (cererea a fost primită, interpretată și acceptată de către server). Un cod tipic din această categorie este *200 OK* (cererea a fost rezolvată cu succes). Alt exemplu este *204 No Content* (serverul a rezolvat cererea, dar nu are ce returna clientului).
– Coduri de redirecționare (*3xx*) – indică o redirecționare a cererii spre altă locație ori alt server. Drept exemple, menționăm codurile *301 Moved Permanently* (resursa solicitată a fost asociată unui URI nou și orice referință viitoare la ea trebuie să se realizeze prin acest URI furnizat) și *302 Moved Temporarily* (resursa cerută are asociat un alt URI, dar pentru o perioadă temporară).
– Coduri de eroare provocate de client (*4xx*) – specifică apariția unei erori pe partea clientului (fie cererea este incorectă din punct de vedere sintactic, fie nu poate fi satisfăcută din varii motive). Ca exemple, furnizăm *401 Unauthorized* (cererea necesită autentificarea utilizatorului – *e.g.*, via un nume de cont urmat de o parolă) și *403 Forbidden* (serverul a recepționat o cerere corectă, dar refuză să o satisfacă din diverse motive legate de restricționarea accesului). Un alt cod tipic, des întâlnit, este *404 Not found* (serverul nu găsește resursa specificată).
– Coduri de eroare generate de server (*5xx*) – desemnează coduri semnificând o eroare pe partea serverului (cererea este aparent corectă, dar serverul nu o poate îndeplini din anumite motive). Drept exemplificare menționăm *503 Service Unavailable* (serverul Web nu poate satisface cererea – *e.g.*, din cauza supraîncărcării temporare sau din rațiuni de administrare).

Lista tuturor codurilor de stare este disponibilă în specificațiile HTTP.

În tabelul de mai jos poate fi parcursă lista unora dintre cele mai importante atribute care însoțesc mesajele HTTP.

Atribut HTTP	Semnificație
Accept	Specific unei cereri, cu rol în stabilirea tipului conținutului prin intermediul unei negocieri conduse de server; clientul are posibilitatea de a furniza tipurile media (MIME) pe care acesta le recunoaște și le poate interpreta sau poate indica numai tipul de răspunsuri preferate. Un exemplu este `Accept: text/xml, application/xml`
Cache-Control	Permite controlul *cache*-ului, de cele mai multe ori la nivelul *proxy*-ului dintre client și server – `Cache-Control: max-age=600`
Connection	Atribut general folosit pentru a specifica anumite proprietăți legate de conexiune. Se aplică doar comunicației între două aplicații HTTP din lanțul unor cereri sau răspunsuri. E util în implementarea conexiunilor persistente – `Connection: close`
Content-Type	Desemnează tipul MIME al reprezentării resursei solicitate de un client ori transmise de server. Un exemplu notoriu este `Content-Type: text/html`
Host	Folosit într-o cerere pentru specificarea adresei Internet și a portului în vederea stabilirii exacte a locației unde se află resursa căreia i se adresează cererea – `Host: www.portocale.info`
Location	Specific răspunsurilor HTTP. Utilizat în conjuncție cu coduri de stare de tip *3xx* sau *201 Created*. Poate stabili locația curentă sau preferată a resursei sub forma unui URI absolut.
Server	Conține informații despre aplicația server, cum ar fi numele, versiunea, producătorul, aplicațiile compatibile – `Server: libwww-perl-daemon/1.36`

2. Arhitectura orientată spre servicii Web

2.1. *Introducere*

Sistemul pe care rulează un server Web și care găzduiește o serie de pagini (documente) WWW înrudite – ale unei organizații, companii sau persoane – se numește *sit* (*site*). Această colecție este orientată, de obicei, către anumite informații unitare sau scopuri comune.

Din punctul de vedere al vizibilității, un sit Web poate fi disponibil doar în cadrul unui *intranet* (rețeaua internă proprie unei companii sau organizații) și/sau

în *extranet* (extindere a facilităților intranetului prin mijlocirea comunicațiilor între intraneturile a două sau mai multe organizații).

O *aplicație Web* reprezintă o colecție interconectată de pagini Web cu un conținut dinamic, menită să ofere o funcționalitate specifică utilizatorilor. Natural, interacțiunea dintre aplicație și utilizatori are loc prin intermediul unei interfețe Web. Deseori, termenii de sit și aplicație Web sunt folosiți unul în locul celuilalt, pentru a desemna același concept. Drept exemple de aplicații Web pot fi enumerate *Amazon*, *Basecamp*, *eBay*, *Expedia*, *Flickr*, *Google Maps*, *PHPMyAdmin*, *Wikipedia*, *Yahoo!* și multe altele.

2.2. *Programarea aplicațiilor Web*

Devine evident faptul că trebuie recurs la un mecanism de generare dinamică a conținutului Web redat utilizatorului, prin intermediul browser-ului, în funcție de datele de intrare, de modul de navigare printr-un sit și de diverși alți factori (autentificare, integrare de conținuturi preluate de pe alte situri, preferințe etc.). Informațiile puse la dispoziție pot fi eterogene, provenind din surse de date multiple (fișiere text, baze de date, documente XML, *stream*-uri multimedia, arhive software, rezultate ale unor operații efectuate la distanță, pe alte calculatoare, și așa mai departe).

Din punct de vedere istoric, prima metodă de generare dinamică pe server a reprezentărilor unor resurse solicitate de un client Web vizează standardul *de facto* CGI (*Common Gateway Interface*). Principalele dezavantaje sunt cele privitoare la invocarea concurentă a mai multor *script*-uri CGI, problemele survenite fiind asigurarea scalabilității, a integrării cu alte aplicații, persistența conexiunii și contextul invocării (rulării).

Evoluția a continuat cu apariția altor interfețe de programare Web pe partea de server. Astfel, pot fi menționate *mod_perl* pentru Apache, NSAPI (*Netscape Server API*) și ISAPI (*Microsoft Internet Services API*), funcționând intern conform modelului CGI. Ulterior a apărut și tehnologia *servlet*-urilor Java.

De un succes larg se bucură însă mediile și *framework*-urile care facilitează dezvoltarea de aplicații Web, printre cele mai populare numărându-se ASP (*Active Server Pages*), ASP.NET (parte integrantă a *.NET Framework*), PHP (*PHP: Hypertext Prepocessor*) și JSP (*Java Server Pages*). Principalele avantaje ale acestora față de vechiul, dar încă utilizatul CGI, sunt următoarele: suportul pentru sesiuni, asigurarea *load-balancing*-ului, conexiunile persistente cu sistemele de baze de date, suportul pentru *template*-uri de interfață, facilitățile vizând modularitatea, securitatea etc. Desigur, în unele situații pot fi folosite și cadre de lucru adiționale.

2.3. *Servicii Web. Punerea problemei*

Premise

După cum am mai menționat, originile și scopurile Web-ului iau în considerare constituirea unui spațiu de comunicare interumană prin intermediul partajării cunoștințelor și exploatarea puterii computaționale puse la dispoziție de calculatoarele interconectate. Se poate observa cu ușurință faptul că interacțiunea dintre om și Web se rezolvă prin intermediul formularelor și legăturilor hipertext, iar interacțiunea dintre mașini se desfășoară pe Web într-o manieră limitată.

O primă abordare este aceea de a recurge la un mecanism care să ne permită apelarea unor operații menite a fi executate la distanță. Astfel, un program client poate să invoce proceduri (metode, funcționalități) pe alte calculatoare din rețea. Această soluție este oferită de paradigma RPC (*Remote Procedure Call*), pe care o vom prezenta pe scurt în cadrul următoarei secțiuni.

*RPC (*Remote Procedure Call*)*

Mecanismul RPC a apărut cu mai bine de două decenii în urmă în lumea UNIX și a fost/este folosit în construcția de aplicații distribuite pe sisteme eterogene, având la bază tehnologiile Internet.

Paradigma RPC conferă forță modelului client/server și constituie un instrument de programare mai simplu față de abordarea clasică oferită de *socket*-uri. Dacă în mod obișnuit modelul client/server se axează mai mult pe partea de comunicație între procese aflate pe mașini diferite, RPC este mai aproape de proiectarea clasică a aplicațiilor, programatorul focalizându-se pe logica programului și abia la final divizând aplicația în componente și adăugând suportul de comunicare în rețea.

O aplicație RPC va consta dintr-un client și un server, serverul fiind localizat pe mașina care execută procedura. Aplicația client comunică prin rețea – via TCP/IP – cu procedura de pe calculatorul aflat la distanță, transmițând argumentele și recepționând rezultatele. Clientul și serverul se execută ca două procese separate care pot rula pe calculatoare diferite din rețea.

Aceste procese client și server comunică în mod transparent, prin intermediul a două interfețe numite *ciot* (*stub*): va exista deci un *stub* pentru client și altul pentru server. Interfețele implementează protocolul RPC, care specifică modul cum se construiesc și cum se prelucrează mesajele emise între procesele client și server. *Stub*-urile se generează de obicei cu ajutorul unui utilitar, după care se „leagă" de programele client și server. Fișierele *stub* conțin funcții care translatează

(de obicei, fără aportul programatorului) apelurile locale de procedură într-o secvență de apeluri de funcții RPC de rețea. Astfel, din punctul nostru de vedere, totul se consideră a fi un simplu apel local. Clientul „cheamă" procedurile din *stub*-ul său prin care utilizează biblioteca RPC pentru a găsi procesul la distanță, urmând ca apoi să-i transmită cereri. Procesul la distanță „ascultă" rețeaua prin intermediul *stub*-ului propriu. *Stub*-ul serverului realizează invocarea rutinelor dorite cu ajutorul unei interfețe de apel de proceduri locale.

Clientul și serverul trebuie să comunice prin mesaje, utilizând o reprezentare a datelor independentă de calculator și de sistemul de operare. RPC adoptă un format propriu pentru reprezentarea datelor, cunoscut sub numele de XDR (*External Data Representation*) și descris pe larg în documentul RFC 1014. Tipurile standard suportate de XDR sunt cele uzuale din limbajul C (precum *int*, *unsigned int*, *float* sau *void*), plus altele, suplimentare. *Stub*-urile client și server sunt responsabile și cu translatarea în și din acest format. Se permite translatarea tipurilor predefinite din C, precum și a unor tipuri mai complexe, cum ar fi vectorii de lungime variabilă. Operația de (de)codificare se numește *(de)serializare* sau *(un)marshalling*.

O facilitate importantă oferită de RPC este ascunderea în totalitate a procedurilor de rețea în interiorul interfețelor *stub*. Acest aspect conduce la simplificarea programelor client și server, eliminându-se necesitatea de a controla detaliile legate de comunicarea în rețea. Deoarece sistemul RPC încearcă să ascundă detalii legate de rețea, el include de obicei o convenție legată de schimbul de argumente și rezultate între client și server, mărindu-se astfel gradul de portabilitate a aplicațiilor.

Astfel, RPC pune premisele oferirii de servicii (a căror implementare nu trebuie să fie cunoscută de către clientul care dorește să-l folosească), adresele clientului, serverului și numele serviciilor fiind păstrate la nivel simbolic. Un serviciu de rețea este identificat prin portul la care este oferit și unde există un *daemon* (proces care rulează în fundal) așteptând cererile de conectare. Un port în viziunea RPC reprezintă un canal logic de comunicare. Comunicarea cu serviciul se realizează via XDR, vehicularea datelor decurgând la nivel binar, conform unor reguli prestabilite de codificare, independente de platformă.

Există mai multe implementări ale paradigmei RPC. Prima, de referință, a fost oferită de corporația Sun Microsystems, fiind denumită *Open Network Computing RPC* (*ONC RPC*) – aceasta este de altfel cea mai răspândită implementare pentru UNIX/Linux (detalii în RFC 1057). Procedurile la distanță se vor include într-un program la distanță. Un program aflat la distanță reprezintă unitatea software care se va executa pe o mașină aflată la distanță. Fiecare program aflat la distanță corespunde unui server, putând conține un set de una sau mai multe proceduri

la distanță ori date globale. Procedurile pot partaja date comune. De remarcat faptul că argumentele pasate procedurilor la distanță trebuie să fie încapsulate într-o structură (similară cu *struct* din limbajul C) pentru a reduce numărul de argumente transmise procedurii.

Fiecărui program aflat la distanță i se va asocia un identificator unic, stocat pe 32 de biți, iar fiecare procedură componentă (care va fi executată în cadrul acelui program) este numerotată (indexată) secvențial de la 1 la *n*, unde *n* este numărul maxim de proceduri ale acelui program. De asemenea, fiecare program la distanță va fi însoțit de un număr întreg pozitiv desemnând versiunea. Prima versiune a unui program de obicei este 1. Următoarele versiuni vor fi identificate de alte numere, în mod unic. Numerele de versiuni oferă posibilitatea de a schimba detaliile de implementare sau extinderea capabilităților aplicațiilor fără a asocia unui program un identificator diferit.

Mecanismul RPC își găsește utilizări interesante, dintre care, cea mai importantă o reprezintă accesul la sisteme de fișiere la distanță prin NFS (*Network File System*), prezent în orice mediu UNIX și semnificând de fapt un sistem de fișiere distribuit (*distributed file system*).

O alternativă la ONC RPC este mediul de calcul distribuit DCE (*Distributed Computing Environment*), care constituie și fundamentul unor servicii de rețea vitale din Windows 2000/XP.

În anii '90, RPC a continuat să fie utilizat prin abordarea obiectuală ORPC (*Object RPC*), mesajele de cerere și răspuns de la distanță fiind încapsulate de obiecte. Ca descendenți direcți ai ORPC se pot enumera *DCOM* (*Distributed Common Object Model*) și IIOP (*CORBA Internet Inter-ORB Protocol*). Arhitectura Java pune la dispoziție RMI (*Remote Method Invocation*), iar platforma .NET oferă așa-numitul *.NET Remoting*.

Necesitatea unei arhitecturi orientate spre servicii. Caracterizare

Odată cu proliferarea aplicațiilor Web, dezvoltatorii și-au dat seama că nevoile industriei de profil au crescut, dorindu-se existența unor soluții multi-platformă slab-conectate. Acestea conduc la integrarea aplicațiilor, serviciilor și sistemelor, prin folosirea tehnologiilor Web actuale. Această integrare trebuie să se realizeze într-un mod flexibil, *ad hoc*, în funcție de necesități. De asemenea, trebuie atins un grad înalt de performanță prin asigurarea scalabilității și e necesară existența unui suport pentru crearea și exploatarea unor servicii atașabile (*pluggable*) și „inteligente"; *software*-ul fiind văzut în termeni de serviciu și nu drept aplicație mamut („*software as service*").

Aceasta pune premisele constituirii unor ofertanți de servicii de aplicații (*application service provider*). De asemenea, apare necesitatea proiectării și (re)folosirii

unor standarde pentru îndeplinirea cerințelor legate de vehicularea, disponibilitatea, mentenanța, securitatea și regăsirea datelor.

Spațiul Web poate fi considerat din acest punct de vedere ca o tehnologie *middleware*, extensie a unor modele precum CORBA sau DCOM (a se urmări și figura 1). Componentele intermediare (*proxy*) la nivel de client și server pot juca același rol ca interfețele *stub* RPC.

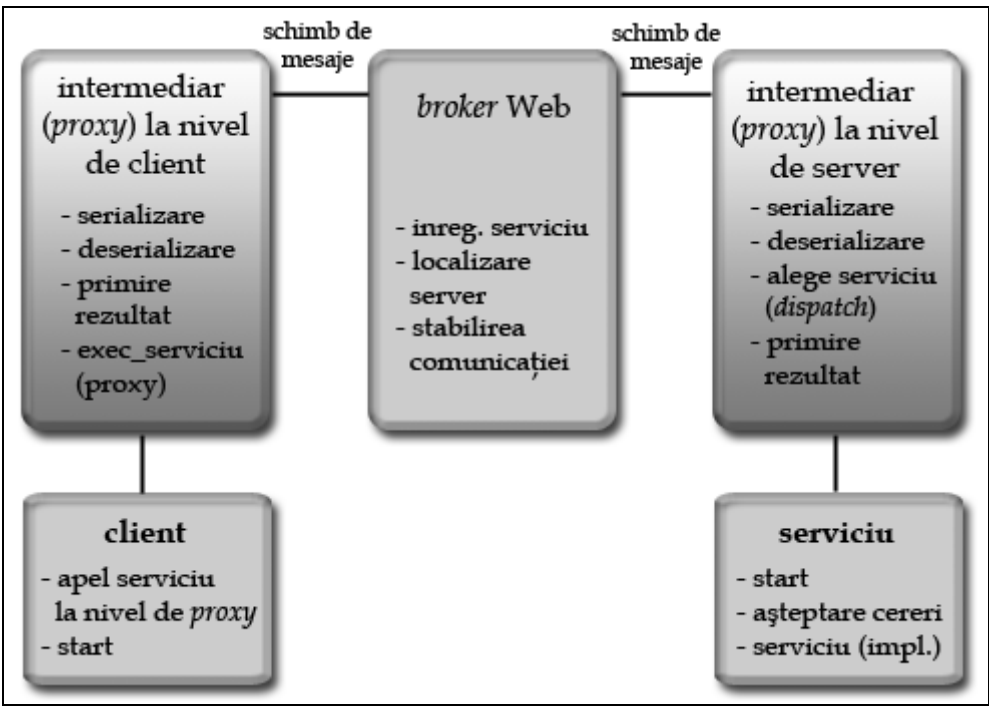


Figura 1. *Spațiul Web ca tehnologie* middleware

Mai mult decât atât, trebuie puse bazele constituirii unei arhitecturi destinate dezvoltării de aplicații distribuite orientate spre Web. Software-ul trebuie divizat în servicii care se pot compune, menite a se conecta și orchestra în mod spontan în cadrul proceselor de afaceri sau din alte medii. Este o viziune a software-ului bazat pe componente Web (*component-based software*), în contrast cu aplicațiile monolitice, clasice. Desigur, software-ul standard („vechi") trebuie integrat în această nouă arhitectură pentru a se proteja investițiile făcute.

Soluția este dată de un model cunoscut sub numele de *arhitectura orientată spre servicii* (SOA – *Service Oriented Architecture*). Arhitectura SOA impune adoptarea unui stil de dezvoltare de aplicații, considerate drept servicii ce vor fi invocate de alte aplicații. Conform OMG (*Object Management Group*), definiția – propusă în aprilie 2006 – a arhitecturii orientată spre servicii este următoarea: SOA reprezintă

un stil arhitectural de obținere a unor valori mutuale de către comunitățile de ofertanți și consumatori de servicii, permițând participanților la aceste comunități de interese să conlucreze într-o manieră cât mai puțin dependentă de tehnologie. Acest stil specifică, de asemenea, contractele pe care organizațiile, oamenii și tehnologiile respective trebuie să le respecte în vederea participării în cadrul comunității, în vederea obținerii de valori de tip *business* și derulării de procese de afaceri. Facilitarea interacțiunilor în cadrul comunității trebuie să se facă prin intermediul unei varietăți de tehnologii (prezente și viitoare).

Conform enciclopediei colaborative deschise Wikipedia, prin SOA se înțelege o perspectivă asupra unei arhitecturi software care definește utilizarea de servicii, oferind funcționalități solicitate de utilizatori. Resursele rețelei sunt astfel disponibile grație unei suite de servicii independente ale căror implementări sunt necunoscute.

SOA presupune că noile servicii pot fi create pe baza celor deja existente, într-o manieră sinergică, dar componentele sistemului în ansamblu trebuie să aibă un grad mare de independență (*de-coupling*). În funcție de schimbările ce pot interveni în cerințe (*business requirements*), serviciile pot fi recompuse sau orchestrate diferit.

Principiile de bază impuse de prevederile SOA sunt:

– serviciile trebuie să partajeze un contract formal;
– serviciile trebuie să fie slab conectate (*loosely coupled*);
– serviciile trebuie să ascundă dezvoltatorului detaliile de implementare;
– serviciile trebuie să ofere suport pentru compunerea cu alte servicii (*composability*);
– serviciile trebuie să poată fi reutilizate;
– serviciile trebuie să se execute într-un mod autonom;
– serviciile nu trebuie să depindă de starea comunicării (*statelessness*), cantitatea de informație specifică unei activități ce trebuie reținută fiind minimală;
– serviciile trebuie să poată fi facil descoperite (*discoverability*).

Astfel, arhitectura SOA trebuie să suporte între aplicații paradigme de comunicare bazate pe Web să ofere o localizare transparentă a serviciilor și să permită adăugarea, înlocuirea și eliminarea serviciilor în mod dinamic.

Prin intermediul SOA, consumatorii de servicii nu sunt dependenți de ofertanții de servicii, putând avea acces la o paletă largă de servicii oferind aceleași funcționalități dorite. Serviciul poate fi văzut doar drept interfață, maniera de procesare (*business logic*) putând fi separată de serviciul propriu-zis.

Via SOA, utilizarea și descoperirea serviciilor se pot realiza în mod dinamic, facilitându-se integrarea la nivel de aplicații (A2A – *Application to Application*)

și/sau de afaceri (B2B – *Business to Business*). Suplimentar, prin intermediul serviciilor se poate realiza managementul proceselor de afaceri (BPM – *Business Process Management*).

Metodologia de proiectare și dezvoltare a aplicațiilor aliniate prevederilor SOA poartă numele de SOAD (*Service-Oriented Analysis and Design*). Una dintre propunerile de astfel de metodologii este cea publicată de IBM în 2004 – SOMA (*Service-Oriented Modelling and Architecture*).

2.4. *Conceptul de serviciu Web bazat pe XML*

Precursori

În mod obișnuit, putem implementa serviciile Web recurgând la *script*-uri CGI sau diverse servere de aplicații. Interacțiunea tradițională poate decurge în următoarele două moduri.

Prima manieră este cea funcțională (de tip cerere/răspuns): utilizatorul (nu neapărat uman) vizitează o pagină și formulează o cerere, fie traversând o legătură hipertext, fie prin intermediul unui formular. Serviciul (implementat de un program conceput într-un anumit limbaj, precum Perl, PHP, C# ori Java) întoarce un răspuns, adică o reprezentare – uzual, marcată în HTML – a resursei solicitate. Astfel, se accesează de la nivelul clientului o anumită funcționalitate specifică pusă la dispoziție de un server Web.

Cea de a doua este una conversațională (de tip solicitare/răspuns). Suplimentar, se dă posibilitatea exprimării unor întrebări adiționale în vederea rafinării cererii: serviciul solicită date de la utilizator pentru a returna un răspuns mai bun.

Se poate observa că serviciul Web, respectând „tradiția", expune o interfață-utilizator (concepută în limbajul de marcare HTML în marea majoritate a cazurilor) prin intermediul căreia are loc interacțiunea. Cererile sunt captate via formulare, utilizatorii umani trebuind să interpreteze etichetele (*labels*) atașate câmpurilor de dialog, pentru a cunoaște ce tipuri de date pot fi introduse (se asigură câmpuri textuale, liste de opțiuni, butoane de tip *checkbox* și *radio* etc.). De asemenea, utilizatorii umani trebuie să interpreteze răspunsul oferit de serviciu prin parcurgerea rezultatului redării de către *browser* a reprezentării (HTML) expediate de server.

Pentru un calculator, activitățile expuse mai sus sunt dificil de realizat, deoarece programul de interpretare a rezultatului depinde de succesiunea de marcaje ale paginii Web recepționate. Orice modificare, minoră sau nu, a *tag*-urilor din cadrul documentului conduce la rescrierea *script*-ului de prelucrare a ieșirii oferite de serverul Web. Această metodă de „capturare" a informațiilor incluse

într-un document HTML se mai numește și *Web/HTML scrapping* și se dovedește dificil de aplicat în cazul receptării de structuri complexe de date.

Definiții și caracterizări

Serviciile Web bazate pe XML (*Extensible Markup Language*) fac explicite specificațiile implicite, putând fi folosite în cadrul interacțiunii dintre mașini. Mai mult decât atât, pot fi invocate și în cazul necunoașterii *a priori* a interacțiunii cu alte aplicații/servicii Web.

Nu există o definiție unanim acceptată a conceptului de serviciu Web. Conform IBM, reprezintă o aplicație modulară bazată pe tehnologiile Internet, menită a îndeplini activități specifice și conformându-se unui format tehnic specific. Microsoft consideră serviciile Web ca fiind partea de procesare a datelor unei aplicații (*application logic*) disponibilă la nivel de program și beneficiind de funcționalitățile oferite de Internet. După Sun, un serviciu Web este o aplicație care acceptă cereri din partea altor sisteme dispuse în Internet sau intranet, mediate de tehnologii de comunicație neutre și agile.

Un serviciu Web oferă o funcționalitate specifică accesată pe baza unui schimb de mesaje marcate în XML. Acest schimb de mesaje e guvernat de anumite reguli. Suplimentar, un serviciu Web se poate autodescrie, folosind meta-date (date referitoare la date).

Serviciile Web implică utilizarea unor protocoale de comunicație, având asociate descrieri ale datelor procesate și alte interacțiuni cu aplicații terțe. Identificarea unui serviciu Web se realizează prin intermediul URI-urilor; transferul de date recurge în mod uzual la HTTP, iar definirea structurată a datelor vehiculate se face via XML. Un serviciu Web poate fi considerat ca fiind compus dintr-o colecție de funcții împachetate, interpretate ca o entitate unică și publicate în spațiul WWW cu scopul de a fi folosite de alte programe.

Ca exemple de operații ce pot fi puse la dispoziție de serviciile Web, le amintim pe următoarele:

– transformarea datelor primite (*e.g.*, conversii de valori în alte unități de măsură);
– dirijarea mesajelor (mai ales în cazul „magistralelor" de date specifice, disponibile la nivel de întreprindere);
– interogări asupra diverselor servere de baze de date relaționale, obiectuale sau native XML (de exemplu, furnizarea cataloagelor de produse, a coordonatelor unor localități, a situației unui cont bancar, a istoricului cumpărăturilor efectuate *online* etc.);
– orchestrarea conversațiilor între diferite entități (jucând, din acest punct de vedere, rolul de mediatori sau agenți);

– realizarea de procesări (calcule) diverse;
– aplicarea de politici de acces asupra resurselor;
– rezolvarea unor excepții ce pot surveni într-un context;
– solicitarea de aprobări de la utilizator (*e.g.*, a execuției unor aplicații de către o altă entitate).

Drept principale caracteristici ale serviciilor Web menționăm:

– accesarea, în manieră publică, se realizează printr-o adresă; serviciile Web pot fi considerate ca reprezentând puncte terminale (*end points*) ale comunicării între mașini, similar modelului folosit de protocoalele de transport ale stivei TCP/IP;
– abilitatea de a prelucra orice tip de date, din moment ce datele de intrare sunt documente XML (conforme unei scheme de validare);
– posibilitățile de dezvoltare pe baza platformelor, arhitecturilor și limbajelor existente;
– adaptabilitatea (un serviciu Web poate fi extins, utilizând eventual, în mod transparent, alte aplicații sau invocând la rândul său alte servicii Web).

Serviciile Web sunt cărămizile pentru crearea de sisteme distribuite deschise, permițând organizațiilor și dezvoltatorilor independenți să-și facă disponibile pe Web bunurile digitale într-un mod rapid și facil.

Mai mult, serviciile Web separă modul de prezentare a datelor de partea de prelucrare a acestora, fiind aliniate șablonului de proiectare MVC (*Model-View-Controller*), des întâlnit în domeniul ingineriei programării. Modulul de interacțiune cu utilizatorul (componenta *view*) este separat de cel de procesare (*model*), reprezentat de serviciul Web în sine, comunicarea fiind facilitată de un strat de control (*controller*) – a se urmări figura 2. Astfel, rezultatele preluate de la serviciul Web, ce oferă o anumită funcționalitate aplicației noastre, pot fi redate utilizatorului într-o multitudine de forme, fără a trebui să modificăm (sau să avem măcar acces la) implementarea serviciului.

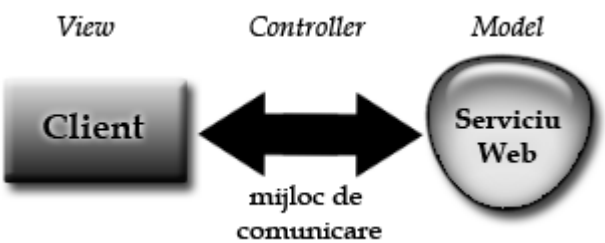


Figura 2. *Interacțiunea între un serviciu Web și un client, prin prisma MVC*

Orice aplicație-client (*script* Perl, *applet* Java, program C cu interfață text sau grafică, aplicație .NET, pagină ori serviciu Web etc.) poate „consuma" – într-o varietate de modalități – datele obținute de la un serviciu invocat printr-o metodă standardizată, bazată pe normele Web în vigoare.

Desigur, în afară de avantajele incontestabile oferite (inter-operabilitate între diverse platforme și limbaje de programare, utilizarea de standarde și protocoale deschise, reutilizarea componentelor software, lipsa unei relații strânse între entități etc.), serviciile Web pot prezenta și dezavantaje, precum lipsa ori suportul precar pentru tranzacții și performanța relativ scăzută față de abordările binare (CORBA, DCOM ori RMI).

Odată cu proliferarea arhitecturii SOA, sunt disponibile servicii specifice mediului *enterprise*, vizând aspecte legate de:

– conținut – specificarea, colectarea și organizarea cunoștințelor din cadrul organizației;
– acces – utilizatorii, via un portal/*desktop* Web, vor avea la îndemână mijloacele de creare și de exploatare a serviciilor sau aplicațiilor compuse;
– integrare – în contextul EAI (*Enterprise Application Integration*) îndeosebi, conținutul vehiculat de aplicații/utilizatori va putea fi transformat grație unor entități software inteligente;
– procesare – se vor putea specifica reguli pentru managementul traficului de date (dirijare, *caching*, filtrare etc.) în funcție de specificul afacerilor întreprinse;
– analizare – acordarea unui suport avansat de luare (inteligentă) a deciziilor;
– colaborare – va putea fi instituită o fundație pentru managementul interacțiunilor și comunicațiilor între utilizatori (umani sau nu) – un exemplu în acest sens este *enterprise wiki*.

XML pentru servicii Web: SOAP, WSDL și UDDI

Vom prezenta în continuare principalele componente folosite în invocarea, definirea și regăsirea serviciilor Web.

În primul rând, apare necesitatea dezvoltării unui *protocol de comunicare* (*transport*) între mașini eterogene care să faciliteze vehicularea de mesaje permițând interacțiunile complexe dintre calculatoare și având un conținut oricât de complex. Mai mult decât atât, trebuie oferit suport pentru asigurarea extensibilității, luându-se în calcul problemele de securitate, fiabilitate și performanță ce pot surveni. Acest protocol va trebui să pună la dispoziție un mecanism de *invocare* și de *transmitere* structurată a datelor.

Trebuie, astfel, găsit un limbaj pentru transmisia în format XML a parametrilor de intrare și întoarcerea răspunsului primit de la serviciul Web invocat.



Figura 3. *Nivelurile de standardizare a serviciilor Web*

Principalele protocoale de comunicare utilizate în prezent sunt XML-RPC și SOAP.

XML-RPC oferă o specificație și un set de implementări pentru realizarea de apeluri de proceduri la distanță, în spiritul mecanismului RPC descris succint mai sus. A fost proiectat pentru a fi cât mai simplu, în același timp însă permițând și transmiterea, procesarea și returnarea unor structuri complexe. În mod succint, putem considera ecuația: XML-RPC = HTTP + XML + RPC.

SOAP (*Simple Object Access Protocol*) reprezintă o recomandare a Consorțiului Web, propunând facilități sofisticate și oferind o paletă largă de posibilități de reprezentare (serializare și deserializare) a datelor vehiculate.

Un mesaj SOAP este compus din trei părți: un *plic* (*envelope*), un set de *reguli de codificare* (*encoding rules*) și o *convenție de reprezentare* (*representation*). Plicul definește cadrul de lucru pentru a descrie ceea ce conține mesajul și modul de procesare a acestui conținut. Datele din corpul mesajului pot fi transportate indiferent de protocol (uzual, se recurge la HTTP). Regulile de codificare sunt de folos la exprimarea instanțelor tipurilor de date definite în aplicație, iar convenția de reprezentare este utilizată în contextul apelurilor de metode implementate de serviciul Web și al răspunsurilor furnizate în urma invocării acestora.

De asemenea, SOAP poate specifica și o cale de la expeditor la destinatar, via un *intermediar* (*proxy*) opțional, în vederea realizării de dirijări de mesaje (*SOAP routing*). Detalii sunt disponibile în capitolul 4 al lucrării de față.

Protocolul SOAP poate fi privit din mai multe perspective. Prima ar fi aceea în care poate reprezenta o extindere obiectuală a paradigmei RPC: cererea și răspunsul conțin parametrii de intrare/ieșire, suplimentar fiind indicate, via XML Schema, tipurile de date ale acestora. A doua consideră SOAP ca fiind un protocol de mesagerie (serializare), cererea incluzând un obiect-cerere serializat, iar răspunsul stocând un obiect-răspuns serializat. Cel de-al treilea punct de vedere – prezent și în cazul REST (vezi *infra*) – admite o tranzacție SOAP ca fiind o transformare XSLT la distanță („*XSLT with a long wire*"): cererea este un document XML, iar serverul returnează, ca rezultat al invocării serviciului, o versiune XML transformată a cererii. Desigur, nici una dintre aceste abordări nu este impusă de protocol.

În vederea exploatării funcționalităților puse la dispoziție de un serviciu Web, trebuie oferită o descriere a acestuia. Apare necesitatea unui *limbaj de descriere*, menit a răspunde la întrebări precum „Care e sintaxa mesajelor vehiculate?", „Cum se desfășoară transferul de date?" și „Cum găsim un serviciu Web?".

Soluția este furnizată de WSDL (*Web Service Description Language*). Limbajul oferă separarea descrierii abstracte a funcționalității oferite de un serviciu de detaliile concrete (de tipul „cum" și „unde" este disponibilă acea funcționalitate). WSDL descrie serviciile Web începând cu mesajele interschimbate între entitatea care oferă și cea care invocă un anumit serviciu. Mesajele sunt specificate în manieră abstractă și apoi sunt asociate unui protocol de rețea, precizându-se de asemenea și un format (o sintaxă). Un mesaj constă dintr-o colecție de date de anumite tipuri, iar schimbul de mesaje este descris ca o operație. Tipurile de date se definesc via scheme XML, la nivel conceptual folosindu-se un model de date reprezentat printr-un set de componente (mesaj, port, manieră de atașare, serviciu) având specificate diverse proprietăți. La nivel sintactic se recurge la XML. Mai multe amănunte privind WSDL vor fi oferite în capitolul 3.

Apare încă o cerință: instituirea unui mecanism pentru publicarea și regăsirea, în manieră publică, a unui serviciu Web. Este posibil ca o anumită funcționalitate oferită de un grup de servicii Web să poată fi regăsită prin intermediul unor interogări formulate de partea care intenționează să folosească acea funcționalitate.

Pentru aceasta s-a constituit un catalog al tuturor furnizorilor de servicii Web publice în vederea regăsirii informațiilor dorite: UDDI (*Universal Description, Discovery, and Integration*), oferind o bază de date distribuită a serviciilor Web din

prisma tipurilor de afaceri electronice pe care acestea le pot modela. Conceptual, datele stocate într-un catalog UDDI pot fi împărțite în următoarele trei categorii:

– paginile albe (*white pages*) semnifică informații generale referitoare la o companie furnizoare de servicii (numele companiei, descrierea tipurilor de afaceri efectuate, adresa etc.);
– paginile aurii (*yellow pages*) includ scheme de clasificare a companiilor și/sau serviciilor disponibile (de exemplu, coduri de clasificare pe criterii geografice, ale categoriei de afaceri, taxonomii referitoare la produsele comercializate și altele);
– paginile verzi (*green pages*) precizează informații tehnice despre un serviciu Web specific (*e.g.*, o adresă Web spre descrierea acestuia, o adresă privitoare la invocarea serviciului etc.).

Uzual, căutarea în cadrul UDDI se realizează în funcție de clasa din care face parte afacerea, după un nume de serviciu sau după locația geografică a furnizorului. Detalii sunt furnizate în capitolul 5.

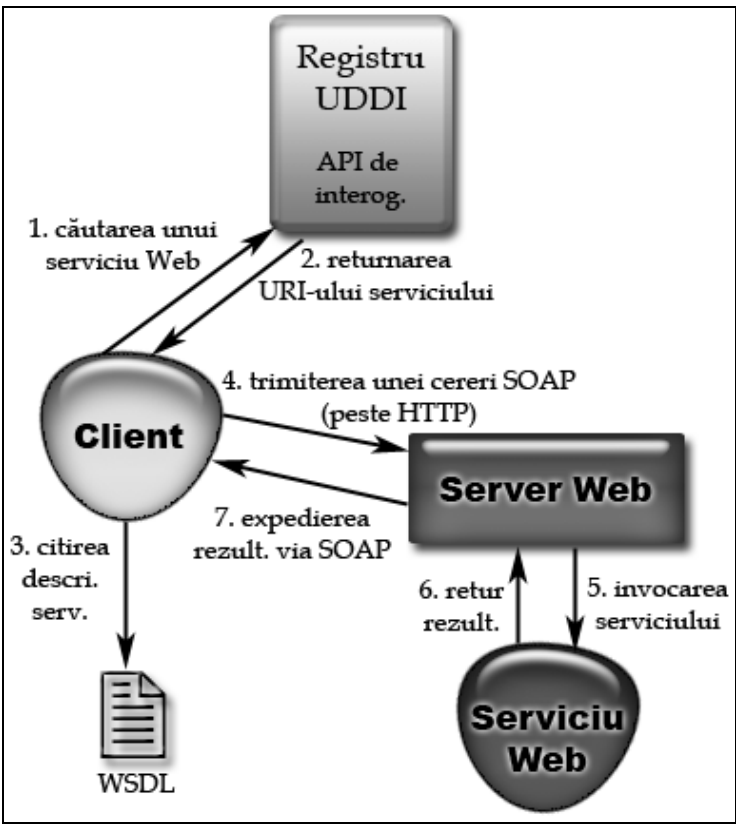


Figura 3. *Relațiile dintre un client, un serviciu Web și registrul UDDI interogat prin SOAP pentru a se obține descrierea WSDL a serviciului dorit*

Funcționalitățile de căutare sunt implementate via servicii Web, astfel încât accesul la registrul UDDI se realizează prin intermediul protocolului SOAP descris mai sus. UDDI va pune la dispoziție documentul WSDL corespunzător unui serviciu compatibil cu cererea formulată – a se vedea și figura 3.

Arhitectura REST

REST (*REpresentational State Transfer*) reprezintă o arhitectură de dezvoltare a aplicațiilor Web. Conform filosofiei REST, rezultatul unei procesări conduce la returnarea unei *reprezentări* a unei resurse Web. Orice accesare a unei reprezentări plasează aplicația-client într-o stare care va fi schimbată în urma unui *transfer* de date (accesarea altei reprezentări, pe baza traversării unei legături hipertext – desemnată de un URI – inclusă în reprezentarea resursei inițiale). Starea comunicării între mesajele vehiculate între server și client nu trebuie reținută (*stateless*).

Transferul de date se realizează prin HTTP, reprezentarea este marcată în XML (ori alt format), iar adresabilitatea se rezolvă via URI, spațiul Web putând fi considerat drept un sistem REST.

Serviciile Web actuale se pot dezvolta în concordanță cu arhitectura REST. Componentele care invocă funcționalități vor consuma reprezentări de resurse (în stilul *pull*) conform clasicei arhitecturi client/server. Fiecare cerere este considerată independentă, fără a se lua în considerație contextul – conexiuni *stateless*. Resursele Web pot fi accesate printr-o interfață generică pusă la dispoziție de metodele protocolului HTTP: GET, POST, PUT și DELETE. Actualmente sunt utilizate preponderent GET și POST.

REST se consideră a fi o viziune complementară de implementare și utilizare a serviciilor Web, în locul mesajelor SOAP (al căror format e considerat de unii dezvoltatori prea complex în anumite situații) recurgându-se la reprezentări XML mai simple – numite și POX (*Plain Old XML*). O astfel de abordare e adoptată și de suita de tehnologii AJAX, prezentată în cadrul capitolului 6.

O aplicație Web dezvoltată conform principiilor REST are o arhitectură diferită de una în stilul RPC (acesta din urmă fiind adoptat și de protocolul SOAP). În cazul RPC, punctul focal e reprezentat de mulțimea de operații ce pot fi executate (numite și verbe), pe când în viziunea REST totul se axează pe diversitatea resurselor disponibile (denumite și substantive).

De exemplu, pentru o aplicație Web privitoare la un sit de comerț electronic oferind sortimente de portocale, abordarea SOAP relevă existența unor operații (metode) precum *furnizeazaSortiment()*, *adaugaSortiment()*, *eliminaSortiment()*, *actualizeaza-Sortiment()*, *listeazaSortimente()*, *cautaSortimente()* – privitoare la sortimentele disponibile –

și *furnizeazaUtilizator()*, *adaugaUtilizator()*, *eliminaUtilizator()*, *listeazaUtilizatori()* și *cautaUtilizatori()* – referitoare la utilizatorii (clienții) magazinului virtual. Toate aceste funcționalități pot fi implementate de un serviciu Web bazat pe SOAP. Recurgând la REST, vom defini doar două tipuri de resurse (*Sortiment* și *Utilizator*), fiecare identificată unic de un URI (de exemplu, *http://www.portocale.info/sortimente/japoneze*). O resursă poate avea asociate reprezentări XML ce pot fi accesate ori alterate. Astfel, prin intermediul unor operații HTTP standard, putem manipula ușor resursele. În acest mod, via GET putem obține o copie a reprezentării unei resurse, cu PUT actualizăm o resursă, iar prin DELETE o ștergem de pe server. Metoda POST se folosește pentru acțiuni care ar putea avea posibile efecte colaterale (*side effects*) – de exemplu, realizarea unei comenzi de plată a sortimentelor de portocale dorite.

După cum se observă, interfața oferită de HTTP este una generică, oferind suport pentru operațiile de bază, de tip CRUD (*Create, Retrieve, Update, Delete*) – creare, accesare, actualizare și ștergere.

Actualmente, există numeroase organizații care își pun la dispoziție serviciile via o interfață REST. Ca exemple notabile, menționăm *Amazon*, *Bloglines*, *del.icio.us*, *eBay*, *Google* și *Yahoo!*. De asemenea, pot fi folosite diverse cadre de lucru pentru dezvoltarea de aplicații Web în stilul REST: JAX-WS (*Java API for XML: Web Services*), *Tonic* (pentru PHP), *Ruby on Rails*, *Zope* (pentru Python) etc.

2.5. *Modalitățile de implementare și exploatare*

Existența serviciilor Web este insuficientă fără a avea la îndemână instrumente suplimentare de implementare, exploatare și integrare. Un aspect important este dat de faptul că informațiile și serviciile trebuie să fie accesibile de pe orice tip de calculator și de oriunde, apărând astfel necesitatea utilizării unei platforme independente de dispozitiv, al cărei rol poate fi jucat de o mașină virtuală.

Noile servicii dezvoltate pot fi compuse din serviciile Web deja existente, accesibile în mod transparent. Ne situăm astfel la nivel de *middleware*, oferind atât funcționalități (implementate de codul-sursă al unor programe concepute în limbaje diverse), cât și suport pentru inter-operabilitate.

Serverele Web actuale trebuie să funcționeze ca porți spre pagini și servicii Web, deoarece încă există o bază largă de situri aliniate stilului „vechi" (*e.g.*, recurgând la *script*-uri CGI și/sau servere de aplicații convenționale).

Soluția este oferită de implementarea și exploatarea unui cadru de lucru (*framework*) Web cu suport pentru SOA. Un astfel de cadru de lucru trebuie să asigure suportul pentru protocoalele Web și cele înrudite (HTTP, SMTP, FTP etc.), plus pentru familia XML.

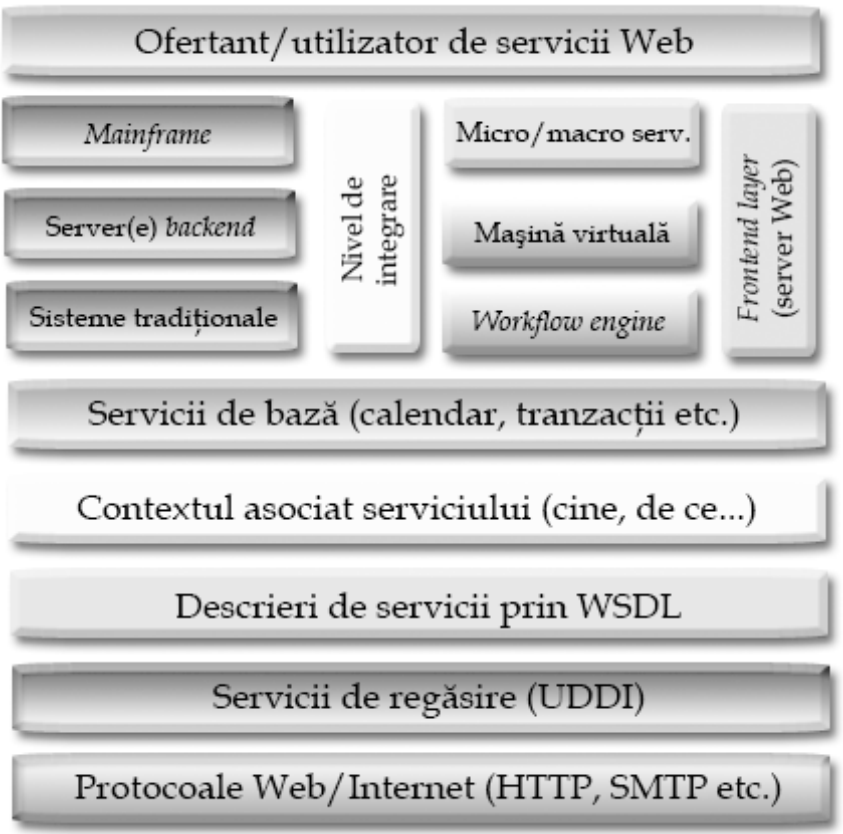


Figura 4. *Arhitectura stratificată a unui cadru de lucru aliniat problematicilor serviciilor Web*

De asemenea, trebuie să se pună la dispoziție mijloace pentru catalogarea, descoperirea și integrarea serviciilor Web via UDDI și descrierea funcționalității și intrărilor/ieșirilor grație limbajului WSDL, cu posibilitatea generării automate de documente WSDL asociate serviciilor Web implementate. Suplimentar, dezvoltatorii vor putea recurge la facilități de specificare și atașare a unui context de execuție, luându-se în calcul factori precum autorizarea (cine poate invoca un anumit serviciu), localizarea (unde e localizat serviciul: platformă, server, port folosit și altele), planificarea (când poate fi rulat serviciul) etc.

Pentru accesul la aplicații și sisteme tradiționale (*legacy*) sau la servere de stocare (*backend servers*), cadrul de lucru va trebui să ofere un nivel de integrare. Mai mult decât atât, e necesară existența unui nivel de interfațare (*frontend*) via un server Web, care presupune existența unei/unor mașini virtuale și a unui motor de asigurare a fluxului de activități (*workflow engine*). La nivelul cel mai de sus se află ofertantul de servicii Web sau utilizatorul direct al acestora – vezi figura 4.

Actualmente există o sumedenie de tehnologii, produse și ofertanți de servicii Web, dintre care le enumerăm doar pe următoarele: *Apache Axis* implementat în Java, *Borland Delphi/JBuilder Web Services*, *IBM Web Services Toolkit*, *JBoss*, *Microsoft .NET Framework*, *NuSOAP* și *PEAR::SOAP* pentru PHP, modulul *SOAP::Lite* pentru Perl, *Web Services Developer Pack* pus la dispoziție de mediul Java. O parte dintre ele vor fi utilizate în cadrul exemplelor incluse în această lucrare, în special în capitolul 6. De asemenea, există implementări oferite și de alte companii precum Apple, BEA, Fujitsu, Hewlett-Packard, Oracle, SAP, Sybase etc.

Mai trebuie să remarcăm faptul că au apărut diverși ofertanți de servicii Web, dintre care îi enumerăm pe *Amazon*, *Blogger*, *cddb* (*CD DataBase*), *eBay*, *European Bioinformatics Institute*, *Google*, *Interfax*, *LiveJournal*, *PayPal*, *RedHat*, *MSN (Virtual Earth)*, *Shopsync*, *Xignite* și *Yahoo!*.

Drept direcții importante de evoluție se remarcă, pe de o parte, implementările bazate pe Java și, pe de altă parte, serviciile Web bazate pe *.NET Framework*.

La finalul acestui capitol notăm că unii specialiști consideră că mediatizarea știrilor (*syndication*) prin formatele deja notorii – RSS (*Really Simple Syndication*) și Atom constituie tot o formă de acces la servicii Web. Astfel, informații diverse – marcate în RSS ori Atom și disponibile prin intermediul serviciilor de tip *feed* – pot fi integrate și prelucrate în diverse alte aplicații de sine stătătoare (*BitTorrent*, *Excel* sau *iTunes*) ori disponibile pe Web (*e.g.*, *Gmail*, *Indeed.com*, *RSSWheather*, *TadaList*, *TagCloud*, *Technorati*, *Yahoo! Maps*).

Referințe

Albin, S., *The Art of Software Architecture: Design Methods and Techniques*, Wiley & Sons, 2003

Alboaie, L, Buraga, S., „Dialoguri despre SOAP", *NET Report*, februarie 2003: http://www.infoiasi.ro/~busaco/publications/articles/SOAP.pdf

Berners-Lee, T., Fielding, R., Masinter, L., *Uniform Resource Identifiers (URI): Generic Syntax*, RFC 2396, Internet Engineering Task Force (IETF), 2004: http://www.ietf.org/rfc/rfc2396.txt

Booth, D. *et al.* (eds.), *Web Services Architecture*, W3C Working Draft, Boston, 2003: http://www.w3.org/TR/ws-arch/

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006: http://www.infoiasi.ro/~busaco/books/xml/

Buraga, S., *Proiectarea siturilor Web* (ediția a doua), Polirom, Iași, 2005: http://www.infoiasi.ro/~design/

Buraga, S., Ciobanu, G., *Atelier de programare în rețele de calculatoare*, Polirom, Iași, 2001: http://www.infoiasi.ro/~lrc/

Cabrera, L.F., Kurt, C., *Web Services Architecture and Its Specifications*, Microsoft Press, 2005

Coyle, F., *XML, Web Services, and the Data Revolution*, Addison-Wesley, 2002

Dumbill, E. *et al.*, *Programming Web Services with XML-RPC*, O'Reilly, 2001

Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, 2005

Fielding, R., „Architectural Styles and the Design of Network-based Software Architectures", *Ph.D. Dissertation*, 2000: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Freed, N., Borenstein, N., *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, Internet Engineering Task Force (IETF), 1996: http://www.ietf.org/rfc/rfc2045.txt

Freed, N., Borenstein, N., *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2046, Internet Engineering Task Force (IETF), 1996: http://www.ietf.org/rfc/rfc2046.txt

Gettys, J. *et al.*, *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Internet Engineering Task Force (IETF), 1999: http://www.ietf.org/rfc/rfc2616.txt

Guruge, A., *Web Services: Theory and Practice*, Digital Press, 2004

He, H., *What is Service-Oriented Architecture?*, XML.com, 2003: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

He, H., *Implementing REST Web Services: Best Practices and Guidelines*, XML.com, 2004: http://www.xml.com/pub/a/2004/08/11/rest.html

Jacobs, I., Walsh, N., *Architecture of the World Wide Web, Volume One*, W3C Recommendation, Boston, 2004: http://www.w3.org/TR/webarch/

Linthicum, D., *Enterprise Application Integration*, Addison-Wesley, 1999

Mahmoud, Q., *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (WAI)*, Sun, 2005: http://java.sun.com/developer/technical/WebServices/soa/

Myerson, J., *The Complete Book of Middleware*, Auerbach Publications, 2002

Tanenbaum, A., *Distributed Operating Systems*, Prentice Hall International, 1995

Weerawarana, S. *et al.*, *Web Services Platform Architecture*, Prentice Hall PTR, 2005

* * *, *Atom*: http://atomenabled.org/

* * *, *Flickr API Documentation*: http://www.flickr.com/services/api/

* * *, *Google Web API*: http://www.google.com/apis/

* * *, *IBM's DeveloperWorks Web Services*: http://www-136.ibm.com/developerworks/webservices

* * *, *MSDN (Microsoft Developer Network)*: http://msdn.microsoft.com/

* * *, „Representation State Transfer", *Wikipedia.org*, 2006: http://en.wikipedia.org/wiki/Representational_State_Transfer

* * *, „Service-oriented architecture", *Wikipedia.org*, 2006: http://en.wikipedia.org/wiki/Service-oriented_architecture

* * *, *Situl lui Paul Prescod*: http://www.prescod.net/

* * *, *Web Services Activity*: http://www.w3.org/2002/ws/

* * *, *World Wide Web Consortium*, Boston, 2006: http://www.w3.org/

* * *, *Yahoo! Developer Network*: http://developer.yahoo.net/

Capitolul 2

Familia XML

„Este mai bine să știi câteva întrebări decât toate răspunsurile."

(James Thurber)

1. Preliminarii

Pentru identificarea resurselor, Web-ul recurge la identificatori uniformi de resurse (URI – *Uniform Resource Identifiers*), iar pentru interacțiune se folosește în mod uzual protocolul HTTP. Reprezentarea resurselor se realizează via formate de date. Arhitectura spațiului WWW încurajează refolosirea formatelor existente, printre aspectele importante legate de acest latură putându-se enumera: adoptarea formatelor textuale în contrast cu cele binare, controlul versiunilor, extensibilitatea, compunerea formatelor, separarea conținutului, prezentării și interacțiunii.

Cu proliferarea serviciilor Internet, mai ales ale Web-ului, datele au putut fi publicate liber, folosindu-se un format de redare (prezentare) a informațiilor pus la dispoziție de bine-cunoscutul HTML (*HyperText Markup Language*), în varianta actuală XHTML (*Extensible HTML*). În prezent, atenția cade asupra modelării cât mai eficiente a informațiilor, prin intermediul unui format deschis, extensibil – subiectul acestui capitol. Mai mult decât atât, modelarea datelor nu reflectă doar sintaxa (care e specificată printr-un set de convenții de marcare), ci și semantica – până acum reprezentată de codul-sursă al programelor ce prelucrau acele date.

Dacă formatele de date transpun maniera efectivă de stocare și prezentare a informațiilor, la nivel conceptual trebuie adoptat cel puțin un model de reprezentare. Evoluția infrastructurilor de calcul a condus și la adoptarea unor modele de date diferite.

Prezentul are la bază arhitecturile navigaționale, bazate pe hipertext (text neliniar), utilizate de o pleiadă de dispozitive, inclusiv cele mobile. Se consideră că unul dintre modelele de date cele mai potrivite este cel oferit de familia de limbaje XML.

2. XML (*Extensible Markup Language*)

Aproape de finalul secolului XX, Consorțiul Web a dorit crearea unui limbaj compatibil cu mai vechiul SGML (*Standard Generalized Markup Language*), dar care să se preteze la utilizarea în cadrul spațiului WWW. În anul 1998, apare prima recomandare-specificație privitoare la XML (*Extensible Markup Language*), iar la momentul scrierii acestei lucrări sunt în vigoare specificațiile XML 1.0 (ediția a patra) și XML 1.1 (ediția secundă).

2.1. *Caracterizare și trăsături esențiale*

Putem considera XML ca reprezentând un standard internațional pentru descrierea de *marcaje* (*markups*) privitoare la resursele electronice. În fapt, XML reprezintă un meta-limbaj (descriere formală a unui limbaj, conform unei gramatici – suită de reguli prin care, pe baza unor simboluri, se generează cuvinte). La început, a fost văzut ca un limbaj de adnotare (de formatare) de texte, dar scopurile actuale depășesc această interpretare îngustă.

Termenul *marcaj* (*markup*) a fost utilizat inițial pentru a descrie anumite adnotări, note marginale în cadrul unui text cu intenția de a indica tehnoredactorului cum trebuie listat un anumit pasaj ori chiar omis. Cum formatarea și imprimarea textelor au fost automatizate, termenul s-a extins pentru a acoperi toate tipurile de coduri de marcare inserate în textele electronice cu scopul de a indica modul de formatare, listare ori alte acțiuni.

Marcajul (*codarea*) reprezintă o acțiune de interpretare explicită a unui fragment de dată. Astfel, un *limbaj de specificare* oferă *un set de convenții de marcare utilizate pentru codificarea textelor*. Un limbaj de marcare trebuie să desemneze mulțimea de marcaje obligatorii, permise, maniera de identificare a marcajelor și care este semantica fiecărui marcaj disponibil (similar procesului de specificare a sintaxei și semanticii unui limbaj de programare).

Există următoarele caracteristici definitorii ale XML, primele trei fiind preluate de la SGML: utilizarea marcajelor descriptive (*descriptive markups*), adoptarea tipurilor de documente via DTD (*Document Type Definition*), independența datelor și caracterul *case-sensitive* (majusculele diferă de minuscule).

Printre trăsăturile care au făcut meta-limbajul XML să fie folosit în industria software putem enumera: suportul acordat Web-ului, facilitățile pentru utilizarea internațională, meta-limbajul (se permite definirea de noi limbaje într-o manieră portabilă), soluțiile pentru reprezentarea conținutului resurselor Web identificate via URI.

Aşadar, XML este o metodă de descriere universală a informaţiei, astfel încât atât computerele, cât mai ales oamenii să o poată înţelege. Scopurile limbajului sunt cele legate de utilizarea lui în Internet, suportând o varietate de aplicaţii, dar fiind mult mai flexibil decât HTML. Oferind o manieră universală pentru reprezentarea (descrierea) informaţiilor hipertext, XML poate fi văzut ca o tehnologie complementară limbajului HTML, nu ca o înlocuire a sa.

2.2. *Părţi componente ale unui document XML*

Un document XML poate cuprinde următoarele categorii de constituenţi: declaraţia, elementele, atributele, entităţile, secţiunile de marcare şi instrucţiunile de procesare.

Documentele XML pot să înceapă cu o declaraţie (prolog) XML care specifică versiunea limbajului XML utilizat – de exemplu: `<?xml version="1.0" encoding="UTF-8" ?>` (de asemenea, s-a precizat şi tipul de codificare a setului de caractere folosit).

Componenta structurală a unui document XML este *elementul*. Diferite tipuri de elemente au asociate nume diferite. Fiecare element trebuie specificat explicit prin intermediul marcajelor (*tag*-urilor). Perechea marcaj de început (*start tag*)--marcaj de sfârşit (*end tag*) este folosită la încadrarea fiecărei instanţe a elementului respectiv în cadrul unui text. În XML se utilizează `<element>` pentru a specifica un *tag* de început şi `</element>` pentru cel de sfârşit, unde `element` este numele unui element oarecare (specificat de utilizator ori de autoritatea care a redactat specificaţiile limbajului bazat pe XML folosit). Regulile sintactice privitoare la numele de elemente sunt similare celor privitoare la identificatorii de variabile. Numele începând cu caracterele „*xml*" (în orice modalitate de scriere, cu majuscule sau minuscule) sunt rezervate. Numele de element nu poate conţine spaţii albe.

Un element poate fi vid (nu conţine nimic între *tag*-urile de început şi sfârşit), putând fi menţionat fie `<element></element>`, fie prin forma prescurtată `<element />`. De asemenea, poate include un text (şir de caractere) ori alte elemente. Mai multe elemente de acelaşi tip pot fi, aşadar, imbricate. Un aspect deosebit de important este cel privitor la faptul că elementele trebuie să fie închise şi imbricate corect.

Conţinutul textual al unui element poate fi compus din caracterele permise de codificarea precizată de atributul `encoding` din declaraţia XML, orice apariţie de spaţii albe multiple fiind implicit redusă la un singur caracter spaţiu.

Pentru a ilustra mai detaliat acest aspect, considerăm un *model structural* foarte simplu. Presupunem că dorim să identificăm o comandă de portocale ce va fi procesată de un sit de comerţ electronic. Astfel avem:

```
<portocale>
  <!-- partea de achiziție de portocale -->
  <achizitii>
     <tip>Portocale grecești fără coajă</tip>
     <cod>P10-01-GR</cod>
     <cantit um="kg">3374</cantit>
  </achizitii>
  <!-- partea de vânzări de portocale -->
  <vanzari>
     <tip>Portocale japoneze albastre</tip>
     <cod>P10-03-JP</cod>
     <cantit um="kg">0107</cantit>
   </vanzari>
</portocale>
```

Exemplul de mai sus nu ne precizează anumite reguli de compunere a comenzii (de exemplu, nu putem impune ca tipul produsului solicitat să nu apară de mai multe ori). Așadar, vom avem nevoie de un mecanism de specificare a structurii.

Un *atribut* este utilizat cu scopul de a descrie o anumită proprietate a unei apariții specifice (particulare) a unui element. Atributele sunt localizate în *tag*-ul de start al unui element, imediat după numele elementului și sunt urmate de „=", apoi de valoarea atributului scrisă între ghilimele sau apostrofuri. Dacă valoarea unui atribut nu este specificată între ghilimele, va fi semnalată o eroare, la fel ca și în cazul în care pentru un atribut nu ar fi atașată și valoarea acestuia. Pentru un element pot exista oricâte atribute, specificate în orice ordine, atât timp cât sunt declarate corect.

Pentru exemplul dat mai sus, am specificat unitatea de măsură „kilogram" prin valoarea „`kg`" a atributului `um` asociat elementului `<cantit>`.

Referințele la entități constituie de fapt pointeri către entități. În XML, entitățile reprezintă unități de text, unde o astfel de unitate poate desemna orice, de la un singur caracter la un întreg document sau chiar o referință la un alt document. O referință la o entitate prezintă construcția sintactică `&nume_entitate;` (caracterul „`&`", urmat de numele entității, apoi de caracterul „`;`"). Una dintre cele mai frecvente utilizări ale referințelor la entități este atunci când se dorește folosirea unor caractere care ar conduce la erori de procesare și deci care nu ar trebui să apară în forma lor normală în text (de exemplu, pentru a genera simbolul „<" vom folosi „`<`"). În momentul în care se întâlnește referința la o entitate în document, aceasta se va substitui cu datele pe care le referă și se va returna documentul cu înlocuirile făcute.

Ocazional, anumite părți ale documentului necesită un tratament special în ceea ce privește modul de procesare. Astfel, există posibilitatea folosirii secțiunii

CDATA (*character data*), cu rolul de inhibare a prelucrării construcțiilor XML. Secțiunile CDATA pot fi inserate oriunde pot apărea datele de tip caracter. Ele sunt utilizate pentru a include blocuri de text conținând caractere care altfel ar fi recunoscute ca marcaje. Acest aspect devine important atunci când documentul include, de exemplu, linii de cod-sursă scrise într-un limbaj de programare care are propriile convenții sintactice.

Sintaxa generală a secțiunii CDATA are forma `<![CDATA[ ... ]]>`. Secțiunile CDATA nu pot fi incluse unele în altele.

Un exemplu este următorul:

```
<achizitii>
   <tip>Portocale greceşti fără coajă</tip>
   <obs><![CDATA[
      Cantitatea trebuie să fie >2000.
   ]]></obs>
</achizitii>
```

Instrucțiunile de procesare reprezintă un tip special de marcaj care conține informații privitoare la anumite aplicații ce urmează a fi executate pentru procesarea conținutului. Un exemplu este instrucțiunea de procesare care permite atașarea de foi de stiluri documentelor XML în vederea redării conținutului acestora:

```
<?xml-stylesheet href="xml2html.xsl" type="text/xsl" ?>
```

2.3. *Membrii constituenți ai familiei XML*

Limbajul XML oferă mai mult decât o sintaxă comodă pentru reprezentarea datelor structurate sau semistructurate. XML consistă dintr-o familie de limbaje bazate pe XML pentru reprezentarea datelor și relațiilor dintre ele.

Această familie de limbaje, menite a adapta conceptele curente de stocare, modelare și publicare pe Web a datelor, este compusă din:

- XML (*Extensible Markup Language*) – subset al specificației SGML, conceput pentru o implementare mai ușoară. Modalitățile de validare sunt concretizate, uzual, de schemele XML, complementare DTD-urilor;
- XLL (*Extensible Linking Language*) – oferind suport pentru specificarea de legături hipertext, concretizat în două componente majore:
  - *XLink* – conceput pentru descrierea legăturilor (simple sau multiple) dintre resursele Web;
  - *XPointer* – are ca scop precizarea în manieră extensibilă a unor scheme de adresare a datelor XML;

- XSL (*Extensible Stylesheet Language*) – permite transformarea documentelor XML în alte tipuri de documente (XML, XHTML sau altele) și atașarea unor obiecte de formatare, în vederea redării conținutului XML în formate precum PDF (*Portable Document Format*);
- *XQuery* – împreună cu limbajul XPath, oferă posibilitatea interogării documentelor XML.

Practic, este imposibil să se enumere toate limbajele bazate pe XML existente la ora actuală, standardizate sau nu. Numeroase limbaje utile, în număr de peste 500, sunt descrise la adresa *http://xml.coverpages.org/*.

2.4. *Spațiile de nume XML*

În unele situații pot apărea confuzii, atunci când se folosesc date din diverse surse (documente) XML care pot avea elemente/atribute cu același nume, dar cu semnificații (semantici) diferite. Pentru evitarea acestor ambiguități sunt folosite spațiile de nume, astfel încât numele de elemente și/sau atribute vor fi identificate în mod unic, evitându-se conflictele.

Necesitatea folosirii spațiilor de nume se poate remarca din exemplul următor, în care considerăm două documente XML, primul cu informații despre partenerii de afaceri, al doilea despre numele furnizorilor de portocale:

```
<!-- parteneri de afaceri -->
<parteneri>
  <partener>
    <nume>Portocal Escu</nume>
    <adresa>Aleea Zăpezilor, 33</adresa>
  </partener>
  ...
</parteneri>

<!-- furnizori -->
<furnizori>
   <furnizor adresa="http://ja.po.ro/">
     <nume>Japo Nez S.A.</nume>
   </furnizor>
   ...
</furnizori>
```

Documentul care le utilizează pe precedentele și folosește spații de nume pentru evitarea ambiguităților („*nume* reprezintă un nume de persoană sau un nume de companie?”; *idem* pentru adresă) ar putea fi următorul:

```
<comanda xmlns:p="http://www.undeva.ro/parteneri/">
  <partener>
    <p:nume>Portocal Escu</p:nume>
    <p:adresa>Aleea Zăpezilor, 33</p:adresa>
  </partener>
  <f:furnizor xmlns:f="urn:furnizori.info"
    f:adresa="http://ja.po.ro/">
    <nume>Japo Nez S.A.</nume>
  </f:furnizor>
</comanda>
```

Atributul `xmlns` este folosit pentru declararea spațiilor de nume, iar valoarea lui trebuie să fie un URI, fie localizând o resursă prin adresa ei – cazul URL (*Uniform Resource Locator*), fie desemnând un nume unic asociat resursei în cauză – facilitate oferită de URN (*Uniform Resource Name*). Prin intermediul construcției `xmlns`, se asociază un *vocabular*, denumit și *spațiu de nume* (*namespace*), pentru elementele în cauză.

2.5. *XML Infoset*

XML nu trebuie considerat a fi doar o manieră de precizare la nivel sintactic a structurii și conținutului unor resurse. Mai mult decât atât, Consorțiul Web a pus problema redactării unei recomandări care să specifice un model de date (abstract) pentru XML, numită *XML Information Set* (sau *XML Infoset*).

Prin intermediul acestei specificații se oferă un punct de vedere comun referitor la: serializarea datelor semistructurate, implementarea/folosirea de interfețe de programare (API-uri) pentru procesarea XML, definirea unor alte recomandări de nivel (mai) înalt.

Acest model de date asigură inter-operabilitatea diferitelor tehnologii, interfețe de programare și aplicații XML, stabilind o interpretare comună a conceptelor folosite.

Sunt definite următoarele concepte de bază, împărtășite de întreaga familie XML:

– *document* (*document information item*) – este considerat a fi un arbore, având nodul rădăcină dat de proprietatea `[document element]`. De asemenea, posedă proprietatea `[children]` oferind o serie de „lucruri" de interes (*items*);
– *element* – specifică un element XML și are atașată proprietatea `[parent]` conținând informații despre elementul părinte căruia îi aparține. Are asociată și proprietatea `[children]` cu semantica descrisă mai sus. Proprietatea `[local name]` specifică numele local al elementului al cărui scop (vizibilitate)

e dat(ă) de proprietatea `[namespace uri]` indicând URI-ul spațiului de nume folosit (vid, dacă nu se specifică spații de nume). Prefixul spațiului de nume utilizat este stocat de proprietatea `[prefix]`. Astfel, se dă posibilitatea precizării calificate a numelor elementelor. Accesul la lista neordonată a atributelor asociate elementului se face via proprietatea `[attributes]`. De asemenea, proprietatea `[namespace attributes]` menționează lista neordonată a atributelor `xmlns` atașate;

– *atribut* (*attribute*) – desemnează conceptul de atribut XML. Numele și spațiul de nume atașat sunt specificate de proprietățile `[local name]` și `[namespace name]`. Elementul căruia îi aparține este indicat de proprietatea `[owner element]`, iar valoarea e specificată de `[normalized value]` (valoarea normalizată, rezultată după expandarea tuturor referințelor la entități);
– *caractere* (*characters*) – corespund informațiilor textuale ale conținuturilor elementelor XML. Proprietatea `[parent]` indică elementul căruia îi aparțin, iar `[children]` conține datele-caracter propriu-zise. Setul de caractere utilizat e oferit de proprietatea `[character code]`.

Prin intermediul XML Infoset, datele XML nu trebuie neapărat să aibă reprezentări textuale, stocate în fișiere *.xml*, ci pot proveni din diverse alte surse (obiecte, rezultate ale unor interogări etc.), cu reprezentări diferite. Intuitiv, structura unui document XML complex poate fi mult mai ușor înțeleasă având asociată o reprezentare grafică arborescentă – a se vedea figura 1.

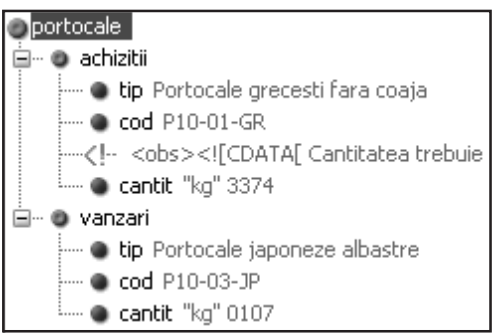


Figura 1. *Reprezentarea arborescentă a unui document XML*

2.6. *Validarea documentelor XML*

Punerea problemei

O primă necesitate în cazul adoptării tehnologiei XML este aceea ca informațiile marcate în XML să poată fi regăsite, reutilizate și partajate între diverse aplicații. Astfel, o cerință importantă este de a cunoaște:
– setul de elemente/atribute ce pot fi specificate;
– modul lor de structurare (*e.g.*, ordinea, numărul minim/maxim de apariții, contextul etc.);
– tipul conținutului (de exemplu, cantitatea de portocale să reprezinte un număr natural, aparținând intervalului [0, 7433]);
– ce anume este valid și ce reprezintă eroare.

Soluția este ca un grup sau grupuri de indivizi (precum o companie, o industrie, persoane, producători de instrumente software specifice sau un consorțiu) să specifice mulțimea de elemente și atribute permise și regulile de marcare, adică tocmai modelul structural amintit mai sus.

Modelul structural se aplică unei clase de documente XML, în vederea verificării printr-un analizor (numit și procesor sau *parser*) a corectitudinii instanțelor de documente aparținând acelei clase. Se au în vedere aspecte privind:
– modul de numire a elementelor/atributelor;
– definirea regulilor de utilizare a acestora;
– specificarea structurii și conținutului;
– precizarea anumitor constrângeri;
– oferirea unui set de convenții de numire.

Apare așadar necesitatea specificării unui set de constrângeri asociate documentelor, astfel încât datele XML să fie verificate dacă sunt valide sau nu din punct de vedere structural sau al tipului conținutului.

Modalitățile de precizare a constrângerilor se pot baza pe: descrieri (*DTD* și *XML Schema*), reguli (*Sablotron*) și șabloane (*RELAX NG*) – detalii în cartea lui Sabin Buraga, *Tehnologii XML*, Polirom, Iași, 2006.

Descriere succintă a XML Schema

• Caracteristici

Deoarece este una dintre cele mai utilizate și versatile maniere de validare a documentelor XML, în cele ce urmează vom prezenta cele mai importante aspecte referitoare la *XML Schema*, recomandarea oficială a Consorțiului Web.

O *schemă* reprezintă o specificație formală a gramaticii asociate unui document XML și reprezintă, în fapt, tot un document XML stocat într-un fișier având extensia *.xsd* (*XML Schema Definition*). O schemă XML definește o *clasă de documente XML* conformându-se unui model structural suplimentar, specificând un sistem de tipuri de date în termenii *infoset*-ului (vezi *supra*). Pentru a putea fi verificată validitatea, o instanță a unei clase de documente XML trebuie să aibă asociată o schemă XML. Aceste aspecte sunt asemănătoare celor de la paradigma obiectuală.

O schemă va specifica modul de apariție și tipurile de date pe care le pot lua valorile construcțiilor XML. Rezultatul obținut în urma unei validări încununate cu succes este numit și PSVI (*Post-Schema Validation Infoset*).

Schemele XML sunt utilizate în multe domenii, dintre care le enumerăm pe următoarele:

– verificarea tipurilor de date în contextul sistemelor de baze de date (relaționale) a mașinilor virtuale (CLR, JVM) etc.;
– serializarea automată a datelor;
– invocarea la distanță a metodelor (RMI – *Remote Method Invocation*, SOAP – *Simple Object Access Protocol*) – a se vedea cele discutate în capitolul 4;
– generarea de cod-sursă;
– implementarea de editoare „inteligente”;
– crearea validatoarelor generale de date (*e.g.*, validarea formularelor electronice).

Construcțiile *XML Schema* trebuie să aparțină spațiului de nume indicat de adresa *http://www.w3.org/2001/XMLSchema*. Atributele privitoare la scheme ce apar în cadrul unei instanțe a unei clase de documente vor proveni din spațiul de nume specificat de *http://www.w3.org/2001/XMLSchema-instance*.

• Construcții de bază

Un document *XML Schema* are ca element-rădăcină `<xsd:schema>`. Definirea sau instanțierea unui element se realizează via `<xsd:element>`, iar în cazul unui atribut prin `<xsd:attribute>`.

Asemănător definirii unor tipuri de date și variabile într-un limbaj de programare, într-o schemă XML fiecare instanță de element trebuie să aparțină unei clase (tip) de elemente. Un element/atribut va avea valori permise ce aparțin unui tip de date (simplu sau complex), specificat prin `<xsd:SimpleType>` și, respectiv, `<xsd:ComplexType>`.

Tipurile simple, predefinite ori derivate din cele predefinite, descriu conținutul datelor textuale. În acest caz, nu se permite ca un element să includă alte elemente și nici să aibă asociate atribute. Tipurile simple pot fi folosite și pentru specificarea conținutului atributelor.

Tipurile complexe descriu datele (semi)structurate. În acest caz, se acceptă ca elementele să includă alte elemente (prin reguli de apariție) și să aibă asociate atribute. Elementele de tip complex vor putea conține:

– declarații de elemente, prin intermediul construcției `<element name="`*`nume`*`" type="`*`tip`*`" />`;
– referințe la elemente *a priori* definite: `<element ref="`*`nume`*`" `*`reguli_aparitie`*`="`*`valori`*`" />`;
– declarații de atribute via `<attribute name="nume" type="tip" />`.

Tipurile complexe nu pot fi folosite în contextul stabilirii tipului valorilor atributelor XML.

Consorțiul Web a pus la dispoziție o paletă largă de tipuri predefinite (primite și derivate). Cele mai utilizate sunt enumerate în continuare:

– numerice: `byte`, `unsignedByte`, `integer`, `positiveInteger`, `negativeInteger`, `int`, `long`, `decimal`, `float`, `double` și altele;
– logice: `boolean`;
– privitoare la dată și timp: `time`, `dateTime`, `duration`, `date`, `gYear`, `gMonth`, `gDay` etc.;
– șiruri de caractere: `string`, `token` și altele;
– adrese Web: `anyURI`.

Ierarhia tipurilor de date *XML Schema* este ilustrată de figura 2.

Pentru codificări ale conținuturilor binare în cadrul documentelor XML se poate folosi tipul `base64Binary`, care oferă o reprezentare a oricăror șiruri de octeți prin intermediul a 65 de caractere specificate de documentul RFC 2045. Mulțimea acestor caractere e compusă din literele minuscule și majuscule, cifre, diverse simboluri și caracterele desemnând spațiile albe.

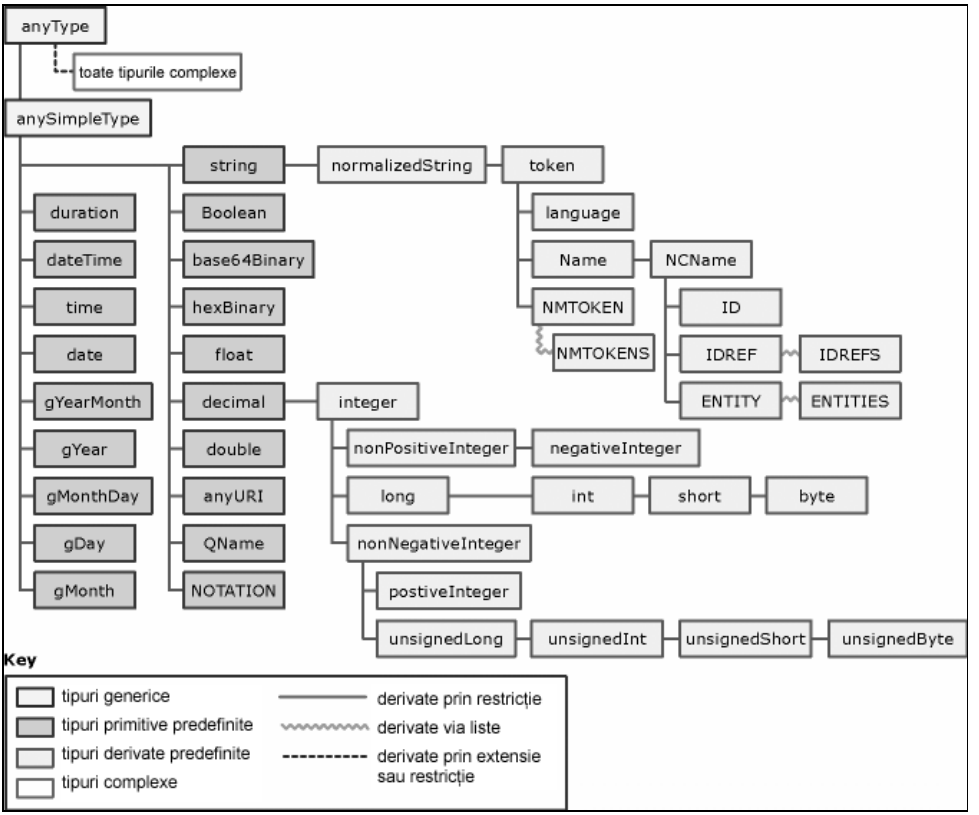


Figura 2. *Tipurile de date specificate de* XML Schema

Putem defini tipuri simple derivate din cele predefinite via `<xsd:simpleType>`. Noul tip de date specificat poate fi o restricție a unui tip deja existent prin intermediul unor constrângeri (*facets*). Prin intermediul constrângerilor pot fi precizate aspecte precum:

– lungimea: `<xsd:length>`;
– lungimea minimă: `<xsd:minLength>`;
– lungimea maximă: `<xsd:maxLength>`;
– un model (*pattern*), exprimat printr-o expresie regulată: `<xsd:pattern>`.

De asemenea, putem recurge la precizarea unei liste de valori (folosind `<xsd:enumeration>`) ce va forma tipul sau a unui interval de valori (construcțiile `<xsd:minInclusive>`, `<xsd:maxInclusive>`, `<xsd:minExclusive>` și `<xsd:maxExclusive>`).

Tipurile simple noi vor fi folosite să descrie valorile elementelor/atributelor, atașarea acestora unor construcții XML putând avea loc fie în cadrul schemei la declararea unui element, fie în cadrul instanței via atributul `xsi:type`.

De asemenea, vom putea preciza următoarele (simbolul „|” semnifică o alternativă):

– reguli (restricții) de apariție a unei instanțe de element: numărul minim de apariții (`minOccurs="`*`număr`*`"`) și/sau numărul maxim de apariții (`maxOccurs="`*`număr`*` | unbounded"`);
– reguli de apariție a unui atribut (obligatoriu, opțional sau interzis): `use="required | optional | prohibited"`;
– valoarea predefinită a unui atribut via `default`;
– valori particulare pentru elemente sau atribute: `fixed`.

De reținut că tipurile simple pot fi utilizate doar pentru a descrie date-caracter. În vederea definirii structurii unui document se recurge la `<xsd:complexType>`. Un tip complex poate avea un conținut simplu (`<xsd:simpleContent>`) sau unul complex.

Conținutul simplu înseamnă că un element va putea include atribute, extinzând astfel modelul-conținut. O primă metodă este derivarea prin extensie, via elementul `<xsd:extension>`, iar a doua modalitate vizează derivarea prin restricție, realizată prin intermediul constrângerilor (*facets*). Atributele vor fi definite prin intermediul `<xsd:attribute>`, putând fi declarate global (la nivelul schemei) sau local (în cadrul unui tip complex). De asemenea, ele pot fi calificate (prefixate de spațiul de nume ales) sau nu.

Specificarea unui tip cu conținut complex vizează definirea listei și ordinii sub-elementelor și atributelor sale. Conținutul unui element poate fi și mixt (compus din sub-elemente sau date-caracter): `<xsd:complexType mixed="true">` ori vid (nu va conține decât declarații de atribute).

Pentru a preciza diverse modele ale conținutului, vom adopta construcții referitoare la:

– alternativă: `<xsd:choice>`;
– secvență: `<xsd:sequence>`;
– grupare: `<xsd:group>`;
– apariție a tuturor elementelor, în orice ordine: `<xsd:all>`.

De menționat și faptul că specificarea unor elemente/atribute generice (ale altor tipuri de documente) se realizează prin elementele `<xsd:any>` și `<xsd:anyAttribute>`. De asemenea, *XML Schema* oferă și suport pentru documentare via `<xsd:annotation>`.

În continuare vom furniza un exemplu de schemă XML menită a valida documentul privitor la achizițiile și vânzările de portocale, menționat în secțiunea 2.2 a capitolului de față. Structura schemei este următoarea:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:portocale.info"
  targetNamespace="urn:portocale.info">
  <xsd:annotation>
     <xsd:documentation xml:lang="ro">
       O schemă utilizată la validarea
       tranzacţiilor de portocale
     </xsd:documentation>
  </xsd:annotation>

  <!-- definirea elementului-rădăcină "portocale" -->
  <xsd:element name="portocale" type="portocaleType" />
  <xsd:complexType name="portocaleType">
     <!-- o secvenţă de alternative -->
     <xsd:sequence maxOccurs="unbounded">
        <xsd:choice>
           <!-- măcar o apariţ. a elem. "achizitii" -->
           <xsd:element name="achizitii"
                type="prodType"
                minOccurs="1" maxOccurs="unbounded" />
            <!-- idem şi pt. "vanzari" -->
            <xsd:element name="vanzari"
                type="prodType"
                minOccurs="1" maxOccurs="unbounded" />
        </xsd:choice>
     <xsd:sequence>
  </xsd:complexType>

  <!-- tipul complex "prodType",
       folosit pentru achiziţii sau vânzări -->
  <xsd:complexType name="prodType">
     <xsd:sequence>
        <xsd:element name="tip" type="xsd:string"
           minOccurs="1" maxOccurs="1" />
        <xsd:element name="cod" type="xsd:string"
           minOccurs="1" maxOccurs="1" />
        <!-- elementul "obs" e opţional -->
        <xsd:element name="obs" type="xsd:string"
           minOccurs="0" maxOccurs="1" />
        <xsd:element name="cantit">
           <xsd:complexType>
              <!-- derivăm dintr-un tip simplu -->
              <xsd:simpleContent>
                 <xsd:extension base="xsd:unsignedInt">
                    <!-- specif. apariţia (obligatorie)
                         a atributului "um" -->
```

```
                        <xsd:attribute name="um"
                            type="xsd:string"
                            use="required" />
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
    <!-- un identificator opțional -->
    <xsd:attribute name="id" type="xsd:ID"
                        use="optional" />
  </xsd:complexType>
</xsd:schema>
```

Reprezentarea grafică a schemei e disponibilă în figura de mai jos.

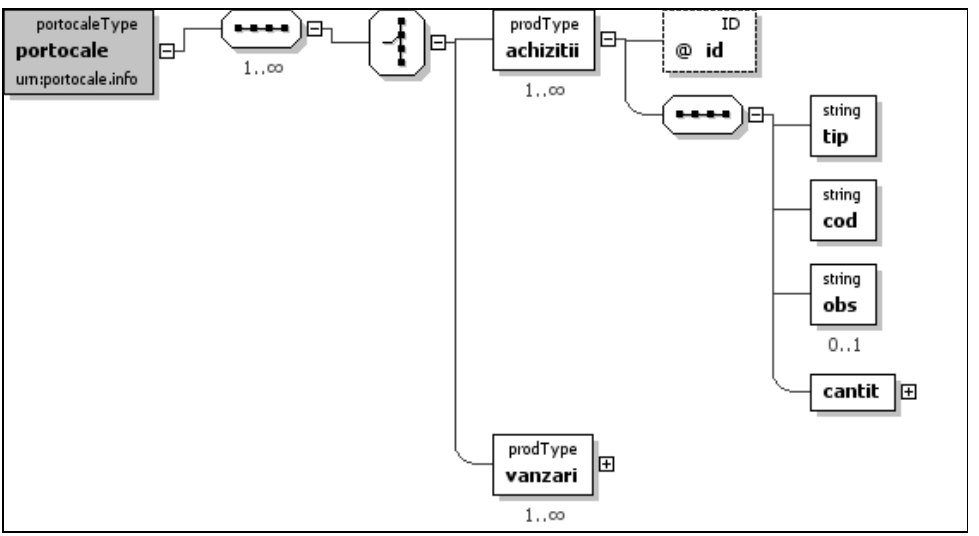


Figura 3. *Reprezentarea generată de editorul <oXygen /> a schemei XML*

Mai urmează să utilizăm schema de mai sus pentru a verifica validitatea unei instanțe de document XML care trebuie să declare un spațiu de nume cu un URI desemnând schema utilizată.

La nivelul unei instanțe de document vom folosi o construcție de genul:

```
<portocale xmlns="urn:portocale.info"
    xmlns:xsi=
      "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
      "urn:portocale.info file:portocale.xsd">
  <!-- conținut propriu-zis -->
</portocale>
```

Aplicația *<oXygen /> XML Editor* amintită mai sus poate fi folosită și ca validator de documente XML.

În cele de mai jos ilustrăm anumite mesaje de eroare afișate de unul dintre utilitarele puse la dispoziție de procesorul *Apache Xerces* – disponibil în regim *open source* – în cazul unui document XML invalid, conform schemei descrise anterior:

```
P:\xerces>DOMPrint.exe -v=always -n -s -f portocale.xml
Error at file "portocale.xml", line 7, column 21
   Message: Unknown element 'suplimentar'
Error at file "portocale.xml", line 11, column 21
   Message: In element cod: Can not have element children
within a simple type content
Error at file "portocale.xml", line 12, column 14
   Message: Required attribute 'um' was not provided
Error at file "portocale.xml", line 13, column 15
   Message: Element 'suplimentar' is not valid for content
model: '((tip,cod,obs),cantit)'
```

Un rezultat aproape similar obținem utilizând *Microsoft Visual Web Developer* pentru editarea și validarea documentelor XML – a se urmări figura alăturată.

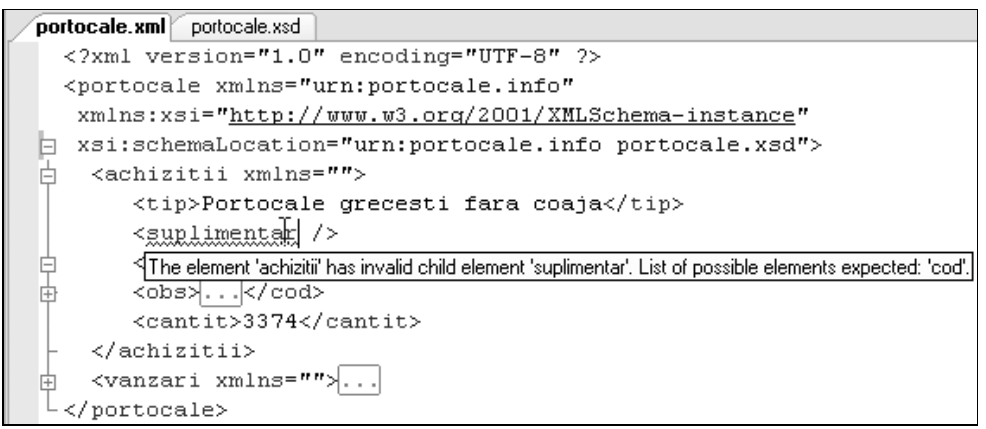


Figura 4. *Semnalarea erorilor de validare a unui document XML în cadrul instrumentului* Visual Web Developer

2.7. *Procesarea documentelor XML*

Modelul DOM

Consorțiul Web a propus pentru prelucrarea sofisticată a documentelor XML și/sau HTML un model obiectual denumit DOM (*Document Object Model*). Acest model reprezintă o interfață de programare a aplicațiilor destinate să prelucreze

documentele HTML și XML, independentă de platformă și de limbaj, definind structura logică a documentelor și modalitățile de accesare și de modificare a lor.

Structura logică a documentelor este o structură arborescentă, conformă cu *XML Infoset*, documentele fiind modelate utilizând obiecte, iar modelul nu furnizează doar o vizualizare structurată a documentului, ci și o modalitate de specificare a comportamentului lui și a obiectelor componente. Fiecare element al unui document poate fi privit, așadar, ca un obiect cu o identitate și o serie de funcții proprii.

Recomandările DOM sunt structurate pe mai multe nivele de specificare a modelului. Nivelul 0 (pentru HTML) a fost nivelul de funcționalitate a versiunilor 3 ale navigatoarelor Netscape și Internet Explorer. Nivelul 1 este recomandarea standardizată din anul 1998, iar în 2000 a fost standardizat DOM – nivelul 2. Parțial, nivelul 3 al DOM a fost publicat ca recomandare oficială în anul 2004 și este în curs de standardizare completă.

DOM nu este o specificație binară și nu definește nici o formă de inter-operabilitate la nivel binar, în contrast cu alte tehnologii, precum CORBA (*Common Object Request Broker Architecture*) ori COM (*Common Object Model*). DOM reprezintă un model care specifică interfețe și nu este un set de structuri de date (abstracte). De asemenea, nu definește semantica detaliată a documentelor HTML sau XML.

Specificația DOM reprezintă documentele ca o ierarhie de obiecte-nod. Anumite tipuri de noduri pot avea noduri copii (descendenți) de diverse tipuri. Altele pot fi noduri frunză, lipsite de descendenți. Tipurile fundamentale de noduri sunt cele din următorul tabel:

Tip	Descendenți
`Document`	`Element`, `ProcessingInstruction`, `Comment`, `DocumentType`
`DocumentFragment`	`Element`, `ProcessingInstruction`, `Comment`, `Text`, `CDATASection`, `EntityReference`
`DocumentType`	-
`EntityReference`	`Element`, `ProcessingInstruction`, `Comment`, `Text`, `CDATASection`, `EntityReference`

Tip	Descendenți
`Element`	`Element`, `Text`, `Comment`, `ProcessingInstruction`, `CDATASection`, `EntityReference`
`Attr`	`Text`, `EntityReference`
`ProcessingInstruction`	-
`Comment`	-
`Text`	-
`CDATASection`	-
`Notation`	-
`Entity`	`Element`, `ProcessingInstruction`, `Comment`, `Text`, `CDATASection`, `EntityReference`

Pentru fiecare tip de nod, DOM oferă o interfață care desemnează constantele, variabilele și metodele ce vor putea fi folosite de programator într-o implementare efectivă a modelului. Există o serie de interfețe fundamentale (*e.g.*, *Document*, *DocumentFragment*, *Node*, *NodeList* sau *Attr*), plus diverse interfețe extinse pentru a suporta implementări având în vedere procesarea documentelor HTML ori oferind facilități adiționale.

În continuare, vom descrie succint o serie dintre interfețele puse la dispoziție.

O interfață importantă este *DocumentFragment* care poate reprezenta un obiect-document minimal. Sunt numeroase situațiile în care nu trebuie lucrat cu întregul document, ci doar cu diverse fragmente ale sale. Arborele de noduri al unui fragment de document este un sub-arbore al structurii de noduri a documentului luat în întregul lui. În funcție de necesități, *DocumentFragment* poate reprezenta o entitate XML, un element XML sau chiar un grup de elemente.

Interfața *Document* reprezintă un document XML, desemnând conceptual rădăcina arborelui de noduri-obiecte ale documentului și oferind accesul la informațiile conținute de acesta. Din moment ce elementele (marcatorii), nodurile de tip text, comentariile, instrucțiunile de procesare nu pot exista în afara contextului unui document, interfața *Document* conține de asemenea metodele necesare pentru a crea aceste categorii de obiecte.

Interfața *Document* are ca membri trei atribute:

i. `doctype` reprezintă declarația tipului de document (DTD) asociată unui document particular.
ii. `implementation` specifică implementarea sau implementările disponibile pentru procesarea documentului.
iii. `documentElement` (de tip *Element*) desemnează nodul-rădăcină de accesare a structurii arborescente a documentului.

Menționăm următoarele metode importante:

- `createElement()` creează un element XML;
- `createTextNode()`, `createComment()`, `createCDATASection()`, `createProcessingInstruction()` vor genera noduri-obiect de tip text, comentariu, secțiune CDATA, respectiv instrucțiune de procesare;
- `createAttribute()` creează un obiect atribut care va fi asociat unui element specificat;
- `getElementById()` va întoarce elementul al cărui atribut *id* se potrivește cu cel furnizat ca argument al acestei metode (în acest mod poate fi selectat exact un anumit element, știind că identificatorul asociat trebuie să fie unic);
- `getElementsByTagName()` va returna o listă ordonată de noduri *NodeList* pentru toate elementele corespunzătoare unui *tag*, ordonarea nodurilor realizându-se prin parcurgerea în preordine a arborelui;

Interfața *Node* definește un tip primar pentru întregul model DOM, reprezentând un anumit nod în cadrul arborelui asociat unui document. Atributele `nodeName`, `nodeValue` și `attributes` sunt introduse ca mecanisme pentru furnizarea informațiilor despre noduri fără conversie de tipuri (vizualizare „simplificată", nu una „orientată-obiect"). Fiecare nod va avea asociată o listă ordonată conținând descendenții săi, plus atribute specificând nodul părinte, primul și ultimul nod copil, dacă există.

Ca metode prezentând interes se pot menționa cele care manipulează nodurile copil:

- `insertBefore()` permite inserarea unui nod înaintea celui curent;
- `replaceChild()` substituie un nod-copil;
- `removeChild()` elimină un nod-copil specificat;
- `appendChild()` adaugă un alt nod-copil;
- `cloneChild()` clonează un anumit nod-copil;
- `hasChildNodes()` întoarce *true* dacă există noduri-copil;
- `hasAttributes()` întoarce *true* dacă nodul are atribute (metodă introdusă în DOM nivelul 2);

- `isSameNode()` întoarce *true* dacă nodul curent e identic cu un altul specificat (metodă oferită de DOM nivelul 3).

NodeList reprezintă o interfață care oferă un tip abstract de dată pentru colecțiile ordonate de noduri, fără a defini sau restricționa cum va fi implementată efectiv această colecție. Fiecare implementator va decide ce tipuri de date concrete vor fi utilizate. Membrii colecției se vor accesa prin metoda `item()` pe baza unui index întreg, numerotarea nodurilor începând cu valoarea 0.

Interfața *NamedNodeMap* este folosită pentru reprezentarea abstractă a colecțiilor neordonate de noduri, prelucrate prin intermediul numelui. Interfața *NamedNodeMap* nu derivă din *NodeList*.

Interfața *Attr* modelează un atribut din cadrul unui obiect de tip *Element*. Tipic, valorile permise ale atributelor sunt definite în schema XML corespunzătoare documentului. Un obiect *Attr* nu se consideră că aparține arborelui de noduri-obiect al documentului. Nodurile de tip *Attr* sunt considerate proprietăți ale elementelor (marcatorilor), putând fi asociate nodurilor *Element* conținute de obiecte de tip *DocumentFragment*.

Interfața *Element* derivă din *Node* și oferă metode de accesare a obiectelor *Attr*, prin nume sau prin valoare: `getAttribute()`, `setAttribute()`, `removeAttribute()`, `getAttributeNode()`, `setAttributeNode()`, `removeAttributeNode()`.

Interfața *Text* este o interfață reprezentând conținutul textual (date de tip șiruri de caractere) al unui nod *Element* sau *Attr*. Dacă între *tag*-urile de început și de sfârșit nu există alți marcatori, atunci textul va fi stocat într-un obiect implementând interfața *Text*.

În prezent există implementări complete pentru DOM – nivelurile 1 și 2 – în majoritatea limbajelor de programare actuale (C/C++, C#, Java, JavaScript, Perl, Python etc.). Drept exemple notabile de biblioteci și interfețe de programare disponibile în diverse limbaje pot fi enumerate *JAXP* (*Java API for XML Parsing*), *JDOM*, *libxml*, *MSXML.NET*, *QDOM*, *Xerces DOM API* și *XML::DOM*.

O serie de exemple de manipulare prin DOM a documentelor XML, pentru procesarea răspunsurilor obținute de la serviciile Web invocate sau în contextul AJAX, sunt furnizate în cadrul capitolului 6.

Procesarea XML prin SAX

Pentru documente de dimensiuni mari, modelul DOM este ineficient, deoarece înainte de a se realiza prelucrarea, documentul respectiv trebuie încărcat complet în memorie. Ca alternativă la implementările DOM, există o interfață simplă de programare destinată manipulării documentelor XML, însă nu atât de completă precum DOM. Această interfață este denumită SAX (*Simple API for XML*).

Majoritatea celor care se ocupă de prelucrarea documentelor XML văd structura unui document în formă arborescentă, iar modelul DOM oferă din plin posibilitatea de a manipula informațiile în această manieră. Din punctul de vedere al implementatorilor această abordare are numeroase deficiențe (maniera de stocare internă a arborelui de obiecte, parcurgerea lui etc.). Unul dintre beneficiile utilizării interfeței SAX este că arborele nu mai trebuie construit, dându-i-se programatorului o altă cale de analiză a documentului XML. Mai mult, SAX poate ajuta la convertirea datelor din formatul arborescent DOM în alt format mai comod, iar pentru procesare nu este necesar a se memora întreaga informație XML, ci numai părțile dorite.

Detalii privitoare la SAX și alte maniere de procesare XML sunt oferite de resursele bibliografice indicate.

Analizoare XML

Desigur, pentru prelucrarea documentelor XML va trebui să recurgem la un analizor XML deja implementat în limbajul nostru preferat. Analizoarele XML pot fi de două tipuri:

– *analizoare fără validare*, care vor procesa un document XML verificând dacă acesta este bine formatat (adică respectă regulile privind sintaxa și modul de imbricare corectă a *tag*-urilor);
– *analizoare cu validare*, care vor realiza procesarea documentului XML prin verificarea regulilor formale descrise de un DTD sau o schemă atașate acestuia (aceste analizoare sunt mai complexe).

Unul dintre cele mai cunoscute analizoare fără validare este *Expat*, disponibil în regim *open source*. De asemenea, pot fi utilizate analizoarele cu posibilități de validare *JAXP* (Sun), *JDOM*, *libxml* (parte a proiectului GNOME), *MSXML* (Microsoft) sau *Xerces* (Apache). Practic toate mediile de dezvoltare Web actuale oferă posibilități de validare a documentelor XML via DTD sau XML Schema și diverse modalități de procesare.

Referințe

Apparao, V. *et al.* (eds.), *Document Object Model (DOM) Level 1 Specification*, W3C Recommendation, Boston, 1998: http://www.w3.org/TR/REC-DOM-Level-1

Biron, P., Malhotra, A. (eds.), *XML Schema Part 2: Datatypes (Second Edition)*, W3C Recommendation, Boston, 2004: http://www.w3.org/TR/xmlschema-2/

Bray, T. *et al.* (eds.), *Extensible Markup Language 1.0 (4th Edition)*, W3C Recommendation, Boston, 2006: http://www.w3.org/TR/xml

Bray, T. *et al.* (eds.), *Extensible Markup Language 1.1 (2nd Edition)*, W3C Recommendation, Boston, 2006: http://www.w3.org/TR/xml11

Bray, T. *et al.* (eds.), *Namespaces in XML (2nd Edition)*, W3C Recommendation, Boston, 2006: http://www.w3.org/TR/REC-xml-names

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006: http://www.infoiasi.ro/~busaco/books/xml/

Buraga, S. *et al.*, *Programare Web în bash și Perl*, Polirom, Iași, 2002: http://www.infoiasi.ro/~cgi/

Cowan, J., Tobin, R. (eds.), *XML Information Set (Second Edition)*, W3C Recommendation, Boston, 2004: http://www.w3.org/TR/xml-infoset

Daum, B., Merten, U., *System Architecture with XML*, Elsevier Science, 2003

Dick, K., *XML: A Manager's Guide* (2nd Edition), Addison Wesley, 2002

Fallside, D., Walmsley, P., *XML Schema Part 0: Primer Second Edition*, W3C Recommendation, Boston, 2004: http://www.w3.org/TR/xmlschema-0/

Geroimenko, V., *Dictionary of XML Technologies and the Semantic Web*, Springer-Verlag, 2004

Harold, E., *XML 1.1 Bible* (3rd Edition), Wiley Publishing, 2004

Le Hors, A. *et al.* (eds.), *Document Object Model (DOM) Level 2 Core Specification*, W3C Recommendation, Boston, 2000: http://www.w3.org/TR/DOM-Level-2-Core

Le Hors, A. *et al.* (eds.), *Document Object Model (DOM) Level 3 Core Specification*, W3C Recommendation, Boston, 2004: http://www.w3.org/TR/DOM-Level-3-Core

Marini, J., *The Document Object Model: Processing Structured Documents*, McGraw-Hill, 2002

McLaughlin, B., *Java & XML* (3rd Edition), O'Reilly, 2006

* * *, *Apache XML*: http://xml.apache.org/

* * *, *Café con Lèche*: http://www.cafeconleche.org/

* * *, *libxml*: http://xmlsoft.org/

* * *, *MSXML*: http://msdn.microsoft.com/xml

* * *, *Oxygen XML Editor*: http://www.oxygenxml.com/

* * *, *Sun's XML Technologies*: http://java.sun.com/xml

* * *, *World Wide Web Consortium*, Boston, 2006: http://www.w3.org/

* * *, *XML Standards Library*: http://xmlstds.xemantics.com/

* * *, *ZVON*: http://www.zvon.org/

Capitolul 3

Descrierea serviciilor Web

„Analfabetul viitorului nu va mai fi cel care nu știe să citească; va fi cel care nu știe să înțeleagă."
(Alvin Toffler)

1. Punerea problemei

Așa cum am menționat în primul capitol, modul de acces la un serviciu Web este specificat printr-un set de documente care conțin definiții, toate acestea constituind *descrierea serviciului*.

Asemenea documente pot fi văzute drept blocuri menite a furniza descrierea unui serviciu Web, incluzând (a se urmări și figura 1):

– definiția unui serviciu = abstractă + concretă (*service definition* = *abstract* + *concrete*);
– descrierea unui serviciu = definiția serviciului + definiții suplimentare (*service description* = *service definition* + *supplementary definitions*).

Să analizăm fiecare dintre termenii de mai sus:

– descrierea interfeței unui serviciu Web este independentă de implementare și este referită de termenul „abstract". Informațiile despre implementarea și localizarea serviciului Web constituie părți „concrete" ale documentului care descrie serviciul. Termenii „abstract" și „concret" au fost introduși cu sensul de mai sus de către Consorțiul Web în specificațiile privind arhitectura serviciilor;
– definiția serviciului (*service definition*) este compusă din definițiile interfeței (abstracte) și definițiile implementării (concrete);
– descrierea unui serviciu (*service description*) constă de obicei doar din definiția serviciului, însă – opțional – ea mai poate conține informații suplimentare (de exemplu, cum este conectat serviciul respectiv de alte servicii).

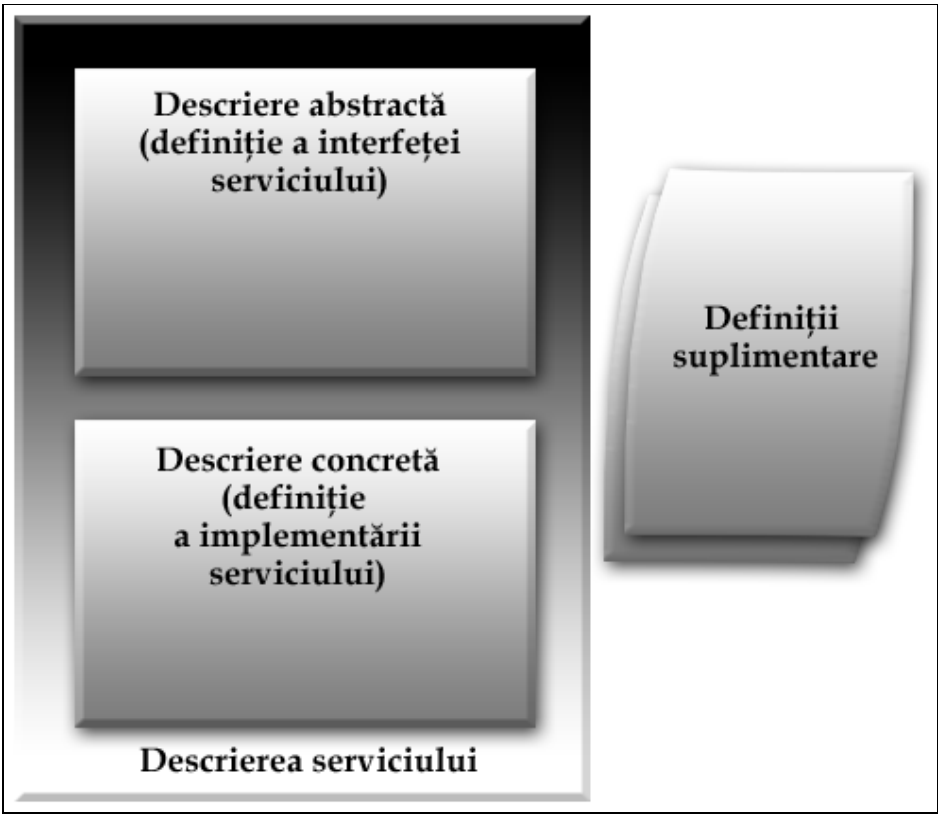


Figura 1. *Conținutul descrierii unui serviciu*

2. WSDL (*Web Service Description Language*)

Serviciile Web trebuie să fie definite într-o manieră consistentă, astfel încât să poată fi ușor descoperite și puse în legătură cu alte servicii și aplicații.

WSDL (*Web Service Description Language*) desemnează o specificație standardizată de Consorțiul Web și este cel mai cunoscut limbaj folosit pentru a descrie un serviciu Web. Prin această descriere se înțelege specificarea locației și descrierea operațiilor (metodelor) permise, aceasta fiind întrucâtva similar modului în care se face descrierea unei componente (interfețe) COM.

Istoric

Specificația WSDL inițială a luat naștere în anul 2000, reprezentând o fuziune a două limbaje de descriere a serviciilor: NASSL (*Network Application Service*

Specification Language) propus de IBM și SDL (*Service Description Language*) provenind din partea Microsoft.

Versiunea actualizată WSDL 1.1 a fost publicată un an mai târziu și trimisă Consorțiului Web spre standardizare. De fapt, formatul WSDL 1.1 a fost larg îmbrățișat de industrie, fiind practic folosit și în prezent.

WSDL 1.2 nu a depășit stadiul de specificație în lucru, fiind compusă din trei documentații (partea de bază, mecanismul de specificare a mesajelor și modul de atașare), ultima actualizare fiind realizată în 2003.

Actualmente, Consorțiul Web lucrează la WSDL versiunea 2.0.

Caracterizare

Conceptual, un document WSDL reprezintă un „contract" între o cerere și un răspuns, similar modului cum o interfață Java reprezintă un „contract" între codul-client și un obiect, instanță a unei clase Java. Diferența esențială este că WSDL poate fi considerat o platformă – și, de asemenea, un limbaj de marcare, care este folosit în principal pentru descrierea serviciilor Web accesate via SOAP.

În figura 2, putem observa cum WSDL, prin intermediul unor descrieri standard, face posibilă comunicarea la nivelul de integrare furnizat de serviciile Web – a se consulta și capitolele 6 și 7.

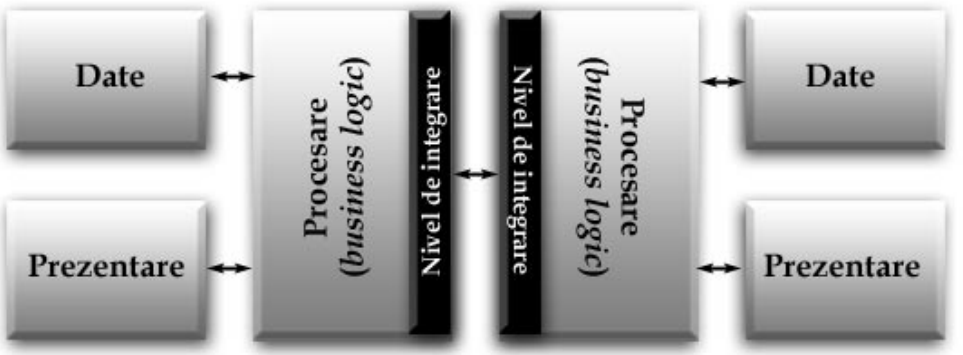


Figura 2. *Maniera de integrare a serviciilor Web via WSDL (documentele WSDL oferă descrierea interfeței de interacțiune cu o aplicație la nivel de integrare software)*

Cea mai potrivită cale de a înțelege cum este descris un serviciu Web cu ajutorul WSDL este de avea o privire de ansamblu asupra modelului său conceptual și de a studia construcțiile care formează această descriere.

2.1. *Modelul conceptual al WSDL*

După cum am văzut, descrierea unui serviciu Web poate fi structurată în două părți principale. În partea abstractă are loc descrierea serviciului Web în termeni de mesaje trimise și recepționate sub controlul unui sistem de tipuri (în mod curent, grație unei scheme XML). Modelul schimbului de mesaje definește succesiunea și cardinalitatea mesajelor. O operație (*operation*) asociază modelul schimbului de mesaje cu unul sau mai multe mesaje. O interfață (*interface*) grupează aceste operații într-un mod independent de transport și de metoda de serializare.

În partea concretă a descrierii, *bindings* specifică modul de transport și formatul de serializare pentru interfețe. Elementul *endpoint* face legătura cu o adresă, iar *service* grupează acele *endpoints* care implementează o interfață comună (SEI – *Service Endpoint Interface*).

Interfața serviciului (definiție abstractă)	Mesaje (*messages*)
	Operații (*operation*)
	Interfață (*interface*)
Implementarea serviciului (definiție concretă)	Legare (*binding*)
	Serviciu (*service*)
	Punct terminal (*endpoint*)

Figura 3. *Modelul conceptual WSDL*

Componentele WSDL

Pentru a descrie un serviciu Web, WSDL furnizează un set de componente și proprietăți asociate acestora.

Avem mai jos forma generală a unui document WSDL:

```
<definitions targetNamespace="xs:anyURI">
  <documentation>?
  [<import> | <include>]*
  <types>?
  [<interface> | <binding> | <service>]*
</definitions>
```

Am folosit următoarele convenții: meta-caracterul „`?`” desemnează o construcție opțională (elementul `<documentation>` poate, așadar, lipsi), caracterele „`[...]`” specifică o grupare de alternative, numele de elemente fiind delimitate de „`|`” (în cazul nostru, poate apărea fie marcatorul `<import>`, fie `<include>`). Simbolul „`*`” specifică zero, una, sau mai multe apariții (elementele `<interface>`, `<binding>` ori `<service>` pot apărea de oricâte ori dorim).

În cele ce urmează vom analiza fiecare element în parte, studiind inițial componentele părții abstracte și apoi pe ale celei concrete.

Elementul-rădăcină al documentului WSDL este *definitions*, putând fi văzut ca un container care conține toate informațiile și atributele asociate unui serviciu Web. Figura următoare arată schema generală a elementului `<definitions>` așa cum este ea redată de editorul *oXygen* pe baza schemei XML oficiale a limbajului WSDL 1.1.

Atributul *targetNamespace* al elementului *definitions* este obligatoriu, de tipul *anyURI*. Spațiul de nume poate defini în mod direct sau indirect semantica documentului WSDL. Marcatorul *definitions* poate avea și alte atribute opționale cărora să le corespundă diferite spații de nume folosite în documentul WSDL. În versiunea 2.0 a limbajului, sub-elementul *portType* este denumit *interface*.

Să considerăm un exemplu concret de utilizare a elementului *definition* pentru specificarea unui serviciu de validare a documentelor XML (implementarea efectivă va fi prezentată în cadrul capitolului 6):

```
<definitions>
  <!-- numele serviciului Web,
       așa cum va fi accesat din exterior -->
  <interface name="XMLValidator">
    ...
  </interface>
  <!-- un mesaj SOAP privind transmiterea
       argumentelor de intrare -->
  <message name="ValidateSoapIn">
    ...
  </message>
  <!-- un mesaj SOAP privind transmiterea
       răspunsului -->
  <message name="ValidateSoapOut">
    ...
  </message>
  <!-- serviciul oferit -->
  <service name="XMLValidator">
    ...
  </service>
  <!-- atașarea (legarea) la protocolul de transport
       efectiv al datelor -->
  <binding name="XMLValidatorSoap">
    ...
  </binding>
  <binding name="XMLValidatorSoap12">
    ...
  </binding>
</definitions>
```

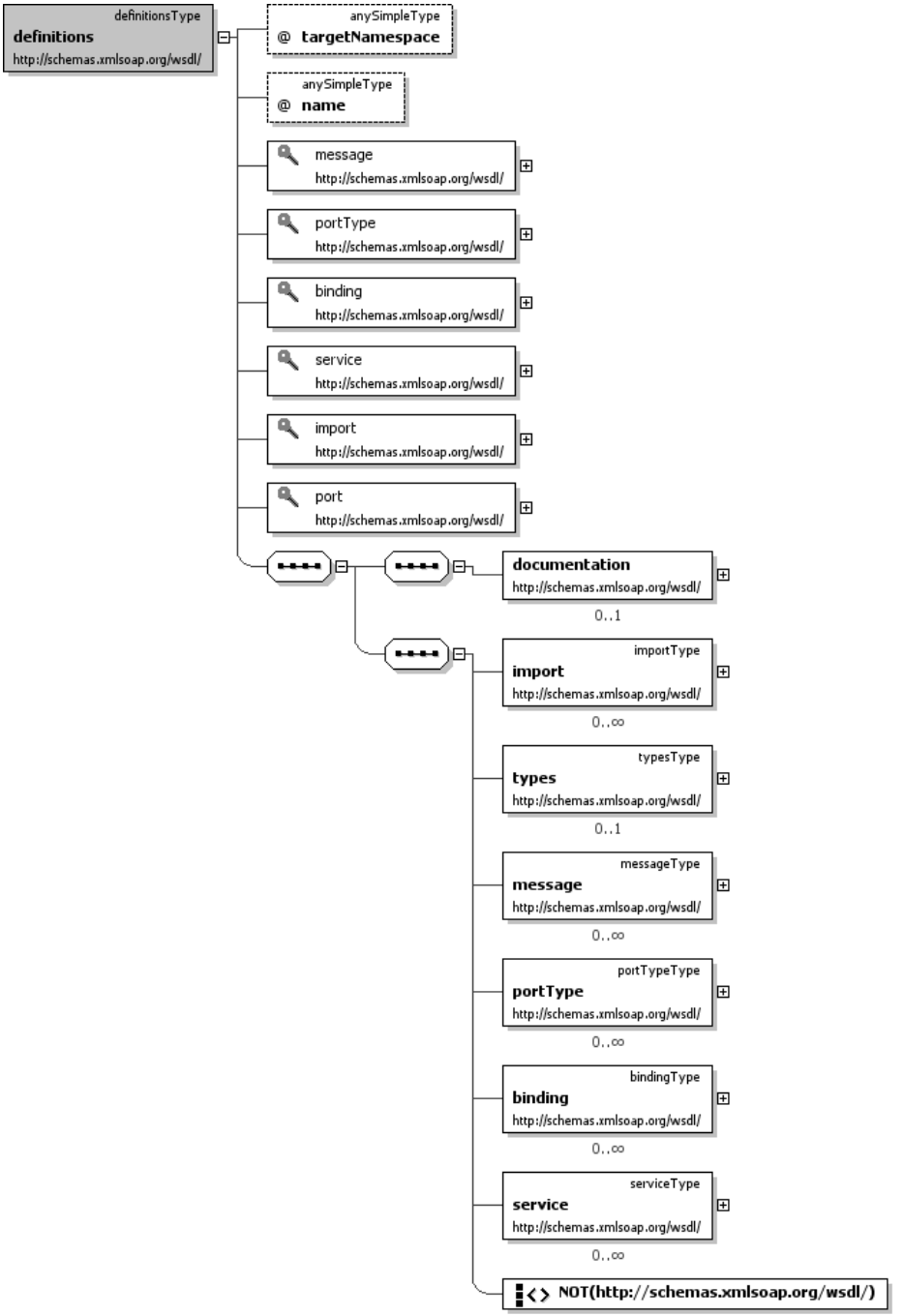


Figura 4. *Schema elementului* definitions

Elementul *interface* include un set de operații abstracte. Opțional, o interfață poate extinde una sau mai multe interfețe. Un exemplu concret de utilizare a elementului *interface* este următorul:

```
<definitions>
  <interface name="XMLValidator">
    <!-- verifică dacă există un fişier -->
    <operation name="CheckIfExists">
      ...
    </operation>
    <!-- validează un document XML -->
    <operation name="Validate">
      ...
    </operation>
  </interface>
</definitions>
```

Elementul *operation* constă dintr-un grup de mesaje de intrare/ieșire legate între ele. Execuția unei operații implică transmiterea sau schimbul unor astfel de mesaje între serviciul consumator și cel furnizor. Elementul *operation* al interfeței are ca atribut obligatoriu *name*, desemnând numele operației expuse.

Reprezentarea acestor mesaje se face cu ajutorul construcției *messages* care se găsește întotdeauna ca element-copil direct al marcatorului *definition*. Numele mesajului este apoi referențiat în operațiile de *input/output* ale elementelor copil (vezi exemplul de mai jos).

```
<definitions>
  <message name="ValidateSoapIn">
    ...
  </message>
  <interface name="XMLValidator">
    <operation name="Validate">
      <input name="msg" message="ValidateSoapIn" />
    </operation>
  </interface>
</definitions>
```

Un element *message* poate conține unul sau mai mulți parametri de intrare/ieșire care aparțin operației. Fiecare element *part* definește un asemenea parametru. El furnizează o pereche nume-valoare împreună cu un tip de data asociat. Exemplul următor ilustrează un bloc *message* având un singur marcator *part*.

```
<definitions>
  <message name="ValidateSoapIn">
    <part name="filename" type="xs:string">
```

```
        document.xml
      </part>
    </message>
</definitions>
```

La nivelul părții de implementare, un document WSDL poate stabili legături concrete cu protocoale cum ar fi SOAP sau HTTP.

Într-un document WSDL, elementul *service* constă din unul sau mai multe elemente *endpoint*, la nivelul cărora serviciul Web poate fi accesat. Aceste puncte terminale includ informații legate de localizare și protocolul permis.

Alte informații specifice protocolului se regăsesc la nivelul elementului *binding*.

Să considerăm mai jos un exemplu de utilizare a elementului *service* (spațiul de nume *tns* este unul temporar):

```
<definitions>
  <service name="XMLValidator">
    <endpoint name="XMLValidatorSoap"
              binding="tns:XMLValidatorSoap">
      <!-- detalii concrete legate de implementare -->
    </endpoint>
  </service>
</definitions>
```

Atributul *name* al elementului *service* este obligatoriu.

Elementul *binding* stabilește protocolul și dă informații despre formatul mesajelor operațiilor. Elementul *operation* din cadrul unui bloc *binding* seamănă cu omologul său din secțiunea *interface*.

Figura următoare oferă forma generală a elementului *binding*.

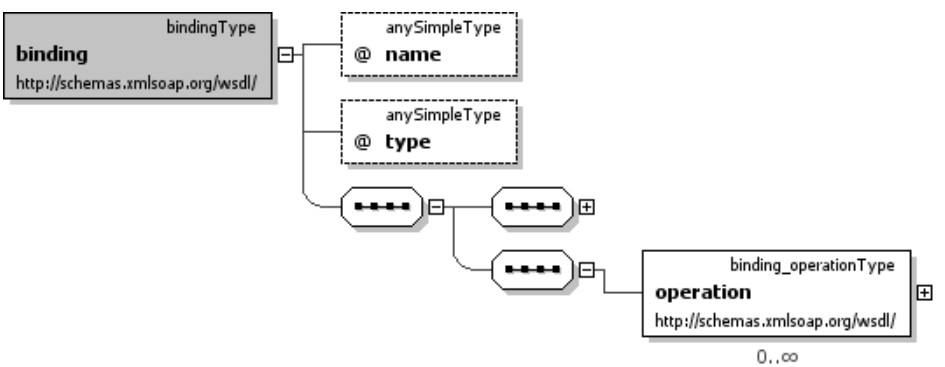


Figura 5. *Elementul* binding

Pornind de la această schemă, un exemplu de utilizare a elementului *binding* este:

```
<definitions>
  <service>
    <binding name="XMLValidatorSoap">
      <operation name="Validate">
        <input>...</input>
        <output>...</output>
      </operation>
    </binding>
  </service>
</definitions>
```

Elementul *feature* asociat elementului *binding* definește funcționalitățile asociate schimbului de mesaje (fiabilitate, securitate, corelație și dirijare). Elementul *property* este folosit pentru a controla marcatorul *feature*. El constă dintr-un set de valori posibile specificate printr-o referință la o schemă de descriere. Aceste valori pot fi partajate între elementele *feature*.

Vom discuta mai departe alte construcții suplimentare pe care le întâlnim în specificațiile WSDL.

Elementul *include* ajută la modularizarea descrierii serviciului Web. El permite ca părți din definiția unor servicii care se referă la același spațiu de nume să poată fi regăsite într-un alt document WSDL, astfel putând fi folosite sau partajate de descrierile serviciilor Web. Atributul *location* este obligatoriu și rolul său este de a specifica locația unor astfel de documente WSDL. Valoarea spațiului de nume asociat documentului WSDL inclus trebuie să fie referită de spațiul de nume al elementului *definition* din documentul WSDL inclus.

Conceptul din spatele elementului *import* este cumva similar celui privitor la *include,* diferența constând în faptul că documentul WSDL importat poate referi un alt spațiu de nume. Atributul *namespace* al marcatorului *import* este obligatoriu, în vreme ce atributul *location* are caracter opțional.

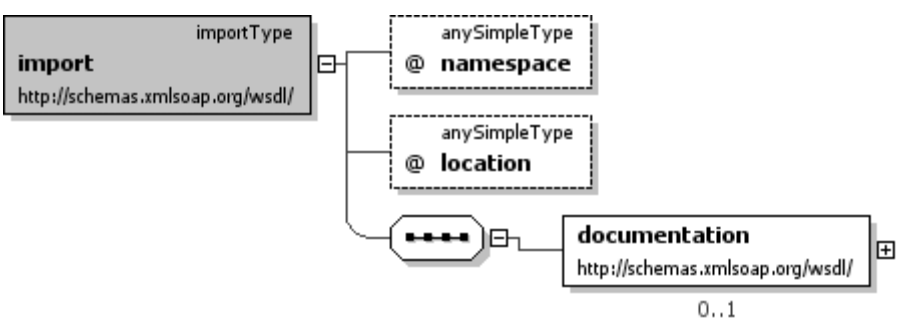


Figura 6. *Schema elementului* import

Elementul *types* definește tipurile de date folosite în comunicare. WSDL nu este legat exclusiv de un anumit sistem care conține aceste tipuri, dar implicit folosește specificațiile XML Schema. Dacă serviciul utilizează doar tipuri simple (precum șiruri de caractere și întregi), atunci elementul *types* nu este obligatoriu.

Un exemplu de utilizare a elementului *types* care stabilește utilizarea tipurilor de date XML Schema este următorul:

```
<definitions>
  <types>
    <xsd:schema
      elementFormDefault="qualified"
      targetNamespace=
        "http://www.infoiasi.ro/XMLValidator"
      xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
      ...
    </xsd:schema>
  </types>
</definitions>
```

Un alt element pe care-l regăsim în forma generală a unui document WSDL este *documentation***.** Acesta poate fi inclus în orice document WSDL și este folosit în special pentru a furniza o explicație textuală ușor de înțeles (*human-readable*).

```
<definitions>
  <documentation>
    Acest serviciu oferă operații de validare XML.
  </documentation>
</definitions>
```

Tipuri de documente WSDL

Sintetizând cele detaliate mai sus, documentele WSDL pot exprima două tipuri de informații: cele referitoare la interfața serviciului și cele privitoare la implementarea efectivă a acestuia.

După cum am văzut, interfața unui serviciu este descrisă de un document WSDL care conține elementele *types*, *import*, *message*, *interface* și *binding*. Interfața unui serviciu conține definiția WSDL a acestuia, folosită pentru implementarea unuia sau a mai multor servicii. Este o definiție abstractă a serviciului Web și este utilizată pentru a descrie un anumit tip (o clasă) de serviciu. Un document de tip interfață poate referi un alt document de tip interfață, folosind elementul *import*.

Un document WSDL de implementare a serviciului conține elementele *import* și *service*. Un astfel de document specifică o descriere a serviciului care implementează interfața serviciului. Măcar unul dintre elementele *import* va face o referință la

documentul WSDL de tip interfață. Un document de implementare a serviciului poate îngloba referințe la mai multe documente de tip interfață (așa cum o clasă dintr-un limbaj orientat-obiect precum C++ ori Java poate moșteni un număr de una sau mai multe clase).

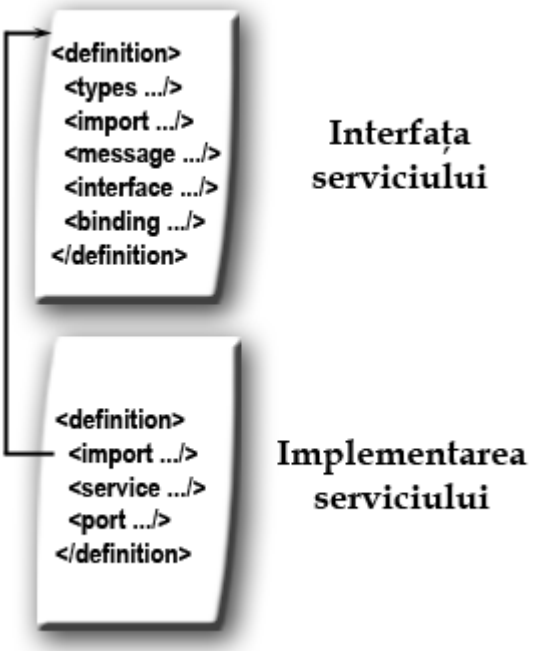


Figura 7. *Tipurile de documente WSDL*

Elementul *import* dintr-un document WSDL de implementare conține două atribute. Primul este *namespace*, a cărui valoare este un URI care se potrivește cu *targetNamespace* din documentul de tip interfață. Al doilea atribut este *location*, precizând un URI folosit pentru a referenția documentul WSDL care conține definiția completă a interfeței serviciului. Atributul *binding* al elementului *port* specifică o referință la un element *binding* particular din documentul de tip interfață.

Documentul de interfață a serviciului este dezvoltat și publicat de un furnizor de servicii de interfață (*service interface provider)*. Documentul de tip implementare este creat și publicat de un furnizor de servicii (*service provider)*. Rolurile acestui furnizor de servicii de interfață și a furnizorului de servicii sunt separate din punct de vedere logic, dar pot desemna și aceeași entitate emițătoare.

2.2. *Diferențele dintre specificațiile WSDL 1.1 și WSDL 2.0*

De la WSDL 1.2 s-a trecut la WSDL 2.0 și întâlnim în principal următoarele diferențe față de WSDL 1.1:

- Adăugarea de semantici limbajului de descriere. De asemenea, s-a introdus atributul obligatoriu *targetNamespace* pentru elementul *definitions*.
- Schimbarea modului de construire a mesajului: se folosește sistemul de tipuri de date din XML Schema via elementul *types*.
- Elementul *portTypes* este redenumit *interfaces*. Suportul pentru moștenirea interfeței este obținut prin folosirea atributului *extends* în cadrul elementului *interface*.
- Elementul *port* a fost redenumit *endpoint*.

2.3. *Exemple de documente WSDL*

În cea mai mare parte, dezvoltatorii de servicii Web nu generează manual descrieri WSDL pentru serviciile lor. Se utilizează anumite instrumente care furnizează documente WSDL în mod automat.

De exemplu, platforma Microsoft .NET poate genera automat o descriere WSDL pentru un serviciu Web. Să considerăm că avem un serviciu Web (vezi capitolul 6 pentru detalii) de salutare a utilizatorului. Pentru a accesa descrierea WSDL generată trebuie să adăugăm sufixul „*?WSDL*" la URI-ul fișierului *.asmx*.

Se va obține o descriere WSDL a serviciului implementat în cadrul fișierului *.asmx* (în exemplul nostru, considerăm serviciul denumit *salutari.asmx*). Facem observația că pentru generarea automată a documentului WSDL de mai jos s-a utilizat *.NET Framework* versiunea 1.1, iar versiunea WSDL folosită este 1.1.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:hs="urn:Hello"
xmlns:soapenc=
  "http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="urn:Hello"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
```

```
  <s:schema elementFormDefault="qualified"
            targetNamespace="urn:Hello">
    <s:element name="sayHello">
        <s:complexType>
           <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                         name="name" type="s:string" />
           </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="sayHelloResponse">
        <s:complexType>
           <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                 name="sayHelloResult"
                 type="s:string" />
           </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="string" nillable="true"
               type="s:string" />
  </s:schema>
</types>
<message name="sayHelloSoapIn">
  <part name="parameters" element="hs:sayHello" />
</message>
<message name="sayHelloSoapOut">
  <part name="parameters"
        element="hs:sayHelloResponse" />
</message>
<portType name="SalutariSoap">
  <operation name="sayHello">
    <documentation xml:lang="ro">
       Metoda de salut sayHello().
    </documentation>
    <input message="hs:sayHelloSoapIn" />
    <output message="hs:sayHelloSoapOut" />
  </operation>
</portType>
<binding name="SalutariSoap" type="hs:SalutariSoap">
  <soap:binding
     transport="http://schemas.xmlsoap.org/soap/http"
                style="document" />
  <operation name="sayHello">
      <soap:operation soapAction="urn:Hello/sayHello"
                      style="document" />
         <input>
```

```
            <soap:body use="literal" />
          </input>
          <output>
            <soap:body use="literal" />
          </output>
    </operation>
  </binding>
  <service name="Hello">
    <documentation>
      Un serviciu Web care vă salută.
    </documentation>
    <port name="HelloSoap" binding="hs:HelloSoap">
      <soap:address
        location="http://localhost/salutari.asmx" />
    </port>
  </service>
  </definitions>
```

În exemplul de mai sus se pot identifica elemente ca *<definition>*, *<message>*, *<portType>* (din versiunea WSDL 1.1), *<binding>* etc.

Reprezentarea grafică a structurilor XML anterioare este dată de figura 8.

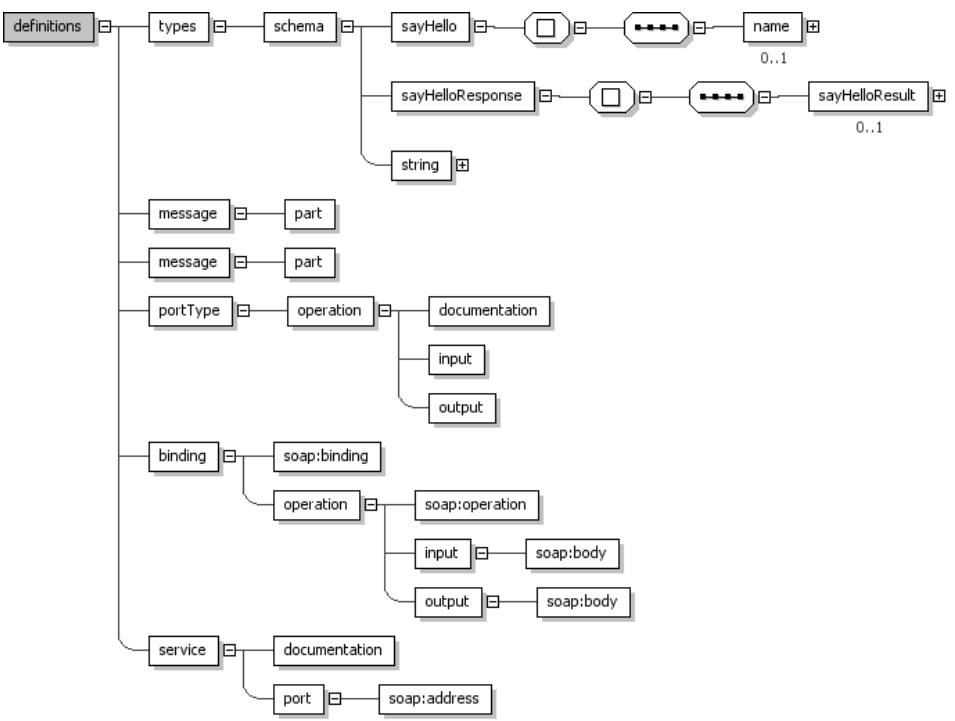


Figura 8. *Reprezentarea grafică a arborelui de elemente corespunzător unui document WSDL*

După cum se poate observa, elementul *definitions* specifică mai multe spații de nume care vor fi utilizate în acest document.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:hs="urn:Hello"
  xmlns:soapenc=
    "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tm=
    "http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="urn:Hello"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Folosirea spațiilor de nume este importantă pentru diferențierea elementelor și permite documentului să facă referințe multiple la specificații externe, incluzând specificații WSDL, SOAP și XML Schema. Atributul *targetNamespace* este un atribut XML Schema care permite documentului WSDL să facă referire la el însuși. Trebuie observat faptul că această valoare trebuie specificată în mod unic (diferită de alte spații de nume definite).

Se remarcă, de asemenea, faptul că elementul *definitions* declară și un spațiu de nume implicit: `xmlns="http://schemas.xmlsoap.org/wsdl/"`. Elementelor care nu au specificat un spațiu de nume (de exemplu, *message* sau *portType*) le corespunde acest spațiu de nume WSDL implicit.

În documentul WSDL sunt definite două elemente *message*.

```
<message name="sayHelloSoapIn">
  <part name="parameters"
        element="hs:sayHello" />
</message>
<message name="sayHelloSoapOut">
  <part name="parameters"
        element="hs:sayHelloResponse" />
</message>
```

Primul reprezintă mesajul-cerere, *sayHelloSoapIn*, iar al doilea desemnează mesajul de răspuns, *sayHelloSoapOut*. Fiecare din aceste mesaje conține un singur element *part*. Pentru cerere, *part* specifică parametrii funcției (metodei), iar pentru răspuns desemnează valoarea întoarsă de aceasta. În exemplul de mai sus, se poate observa că există un singur parametru de intrare pentru care `type="s:string"`. La fel, avem un singur parametru de ieșire tot de tip *string*.

Dacă există situații în care metoda prezintă mai multe argumente sau returnează mai multe valori, atunci se specifică mai multe elemente *part* corespunzătoare.

Marcatorul *portType* definește o singură operație, numită *sayHello*, având un singur mesaj de intrare (*sayHelloSoapIn*) și un unic mesaj de ieșire (*sayHelloSoapOut*).

```
<portType name="SalutariSoap">
  <operation name="sayHello">
    <documentation xml:lang="ro">
       Metoda de salut sayHello().
    </documentation>
    <input message="hs:sayHelloSoapIn" />
    <output message="hs:sayHelloSoapOut" />
  </operation>
</portType>
```

Valoarea atributului *message* trebuie să fie un nume de element XML calificat (de forma `prefix:nume` unde *prefix* reprezintă prefixul spațiului de nume corespunzător elementului). De exemplu, elementul *input* include un atribut `message="hs:sayHelloSoapIn"` – prefixul *hs* se referă la *targetNamespace* definit mai devreme pentru elementul *definitions*.

2.4. *Tipuri de operații și mesaje*

WSDL suportă patru tipuri de operații principale:

– *One-way* – serviciul primește un mesaj. Astfel operația are un singur element *input*.
– *Request-response* (cerere-răspuns) – serviciul primește un mesaj și trimite un răspuns. Operația are un element de *input*, urmat de un element de *output*. Pentru a specifica eventuale erori se poate utiliza elementul opțional *fault*.
– *Solicit-response* (solicitare răspuns) – serviciul trimite un mesaj și primește un răspuns. Operația are un element de *output* urmat de un element de *input*. În același mod se poate utiliza elementul opțional *fault*.
– *Notification* (înștiințare) – serviciul trimite un mesaj. Operația are un singur element de *output*.

Aceste operații sunt prezentate în figura următoare. Modelul cerere-răspuns este cel mai utilizat pentru serviciile SOAP.

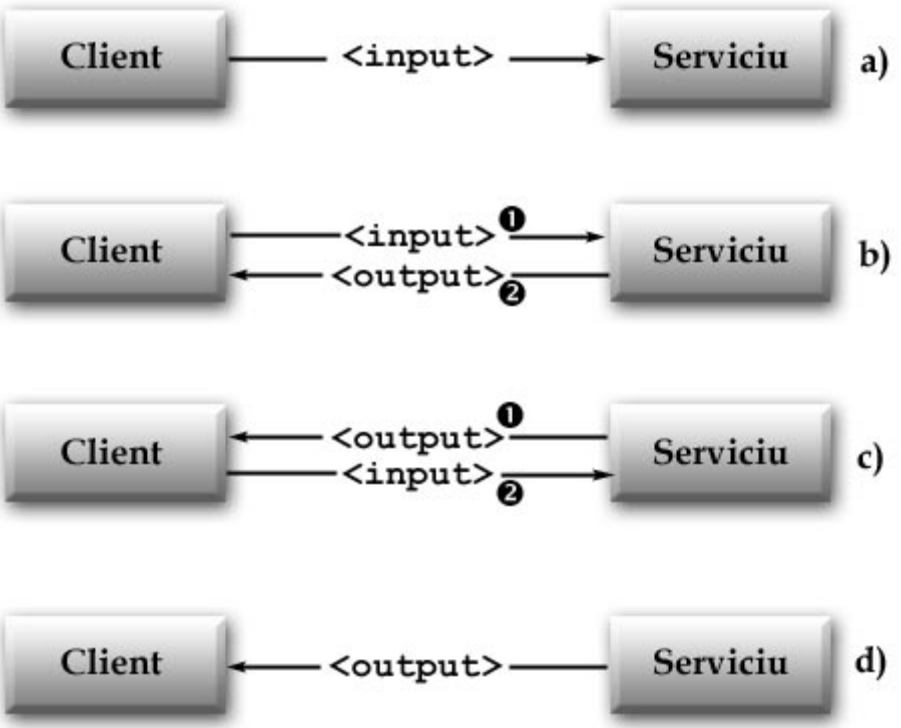


Figura 9. *Tipurile de operații suportate de WSDL:*
a) one-way, b) request-response, c) solicit-response, d) notification

Elementul *binding* furnizează detalii despre modul în care mesajele privitoare la o operație vor fi transmise prin rețea (de exemplu, prin HTTP POST sau SOAP). Elementul *binding* are atributele *name* și *type*:

```
<binding name="SalutariSoap" type="hs:SalutariSoap">
```

Atributul *type* face referință la *portType* definit anterior în document. În cazul nostru, elementul *binding* se referă la *hs:SalutariSoap*. Altfel spus, elementul *binding* afirmă ceva de genul „Voi furniza detalii despre modul cum datele operației *sayHello* vor fi transportate în Internet".

Un document WSDL include construcții SOAP. Aceasta permite specificarea de detalii SOAP, cum ar fi un antet SOAP, *SOAP encoding style*, *SOAPAction* – vezi capitolul următor.

Elementul *soap:binding* indică faptul că legătura cu mecanismul de transport se va face prin SOAP. Marcatorul *soap:operation* desemnează conectarea unei anumite operații la o anumită implementare SOAP. Atributul *soapAction* declară faptul că atributul de antet *SOAPAction* (din cadrul unui mesaj HTTP) este utilizat pentru

identificarea serviciului. Elementul *soap:body* permite specificarea diferitelor detalii despre mesajele de intrare/ieşire.

Marcatorul *service* desemnează locaţia serviciului. Deoarece este un serviciu SOAP, în exemplul de mai sus s-a folosit elementul *soap:address* şi s-a utilizat adresa *http://localhost/salutari.asmx*.

2.5. *Platforme şi instrumente de lucru cu documentele WSDL*

Vom trece în revistă o serie de instrumente de lucru cu WSDL, lăsând la latitudinea cititorului să decidă care este cel mai uşor de utilizat şi mai util pentru fiecare situaţie în parte. Nu vom mai reveni la platforma .NET şi suportul său pentru generarea documentelor WSDL (discutate mai sus), însă remarcăm faptul că este unul dintre instrumentele cele mai utilizate la ora actuală.

Fiind dat un fişier WSDL, se poate crea în mod manual un client SOAP pentru a invoca serviciul. O alternativă mai bună ar fi o invocare automată a serviciului prin folosirea unui instrument WSDL de invocare. Figura de mai jos ilustrează acest mecanism:

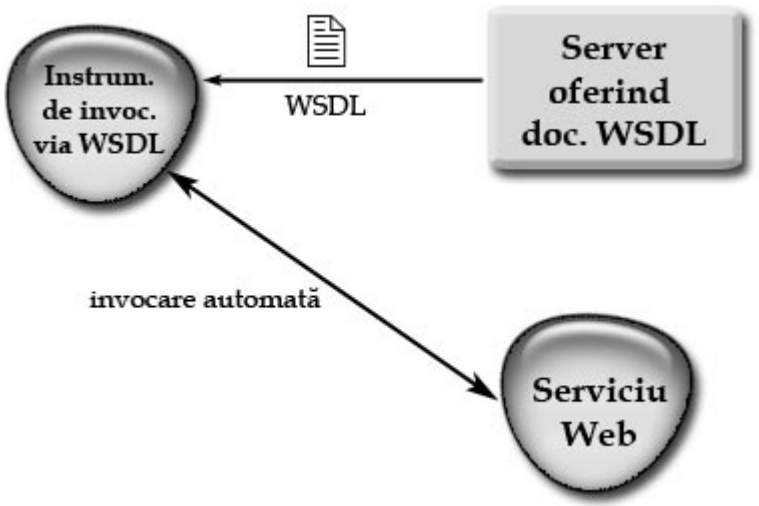


Figura 10. *Invocarea automată a unui serviciu Web via un instrument specializat*

În continuare vom enumera cele mai cunoscute instrumente cu ajutorul cărora se pot face invocări, validări, generări de documente WSDL: WSIF (*Web Services Invocation Framework*), *Stylus Studio's Web Service Call Composer, XML Spy*, *WTP* (însemnând de fapt platforma Eclipse extinsă) şi *SOAEditor*.

Unul din cele mai cunoscute instrumente de invocare este WSIF, proiect iniţiat de către IBM şi donat ulterior iniţiativei *Apache XML – http://xml.apache.org/*.

WSIF este o interfață de programare a aplicațiilor (API – *Application Programming Interface*) folosită pentru invocarea serviciilor descrise cu ajutorul WSDL. Utilizând WSIF, putem realiza o interacțiune cu serviciile Web la un nivel abstract, neținând cont de formatul mesajelor sau de protocolul de transport utilizat.

Poate vă întrebați: la ce este util un astfel de instrument? Să ne imaginăm un sistem software al unei corporații, sistem format din mai multe entități software, dezvoltate într-o perioadă de zeci de ani. Trebuie creată o aplicație care să poată utiliza toate aceste entități. WSIF rezolvă problema folosind descrierile WSDL și permițând accesul la funcționalitățile acestor entități într-o manieră independentă de protocol și de locație. Astfel, indiferent că recurgem la tehnologii ca SOAP (*Simple Object Access Protocol*), EJB (*Enterprise JavaBeans*) sau JMS (*Java Messaging Services*), totul se concentrează asupra unui API dependent de WSDL care permite accesarea tuturor funcționalităților. În consecință, integrarea tuturor componentelor se poate realiza la nivel declarativ, prin intermediul documentelor WSDL.

Stylus Studio's Web Service Call Composer (pe scurt, *Stylus*) reprezintă un sistem care simplifică dezvoltarea serviciilor Web Java, oferind suport pentru găsirea, invocarea și testarea oricărui serviciu Web bazat pe Java, dezvoltat într-un cadru de lucru (*framework*) Java, cum ar fi *Apache Axis*. Vom da mai jos un exemplu al modului în care, cu ajutorul aplicației *Stylus*, putem localiza și invoca metodele unui serviciu Web. Așa cum se poate observa și în figura de mai jos, creăm un proiect de tip *Web Service Call*.

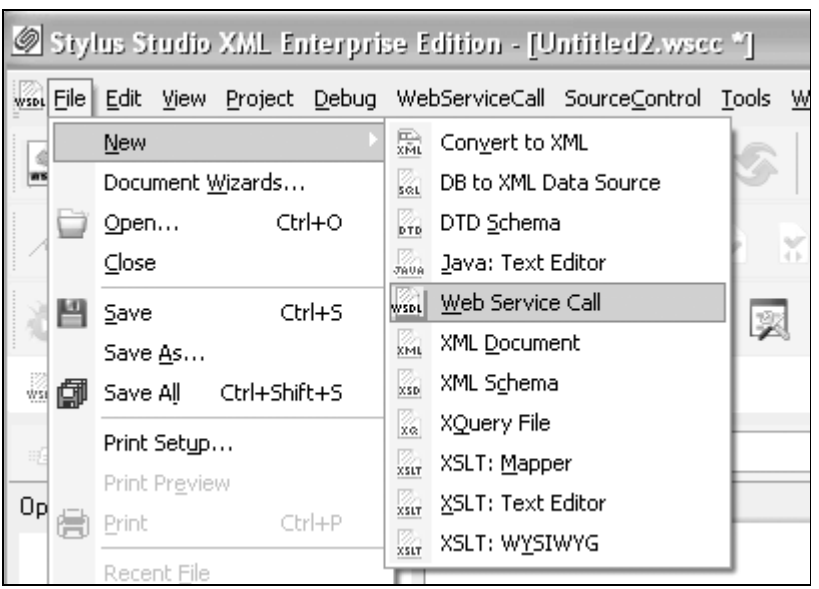


Figura 11. *Crearea unui apel de serviciu Web (*Web Service Call*) prin utilizarea instrumentului* Stylus

Pentru a invoca un serviciu Web, trebuie să putem face o referire la adresa documentului WSDL propriu. Deoarece e posibil ca nu întotdeauna să cunoaștem această adresă, *Stylus* are inclus un browser UDDI care permite căutarea în regiștrii UDDI (vezi capitolul 5). Pentru a realiza o interogare în regiștrii UDDI (de exemplu, IBM, Microsoft, SAP ori XMethods), se alege unul dintre aceștia, se introduce o interogare (de pildă: „*google*") și se pornește căutarea.

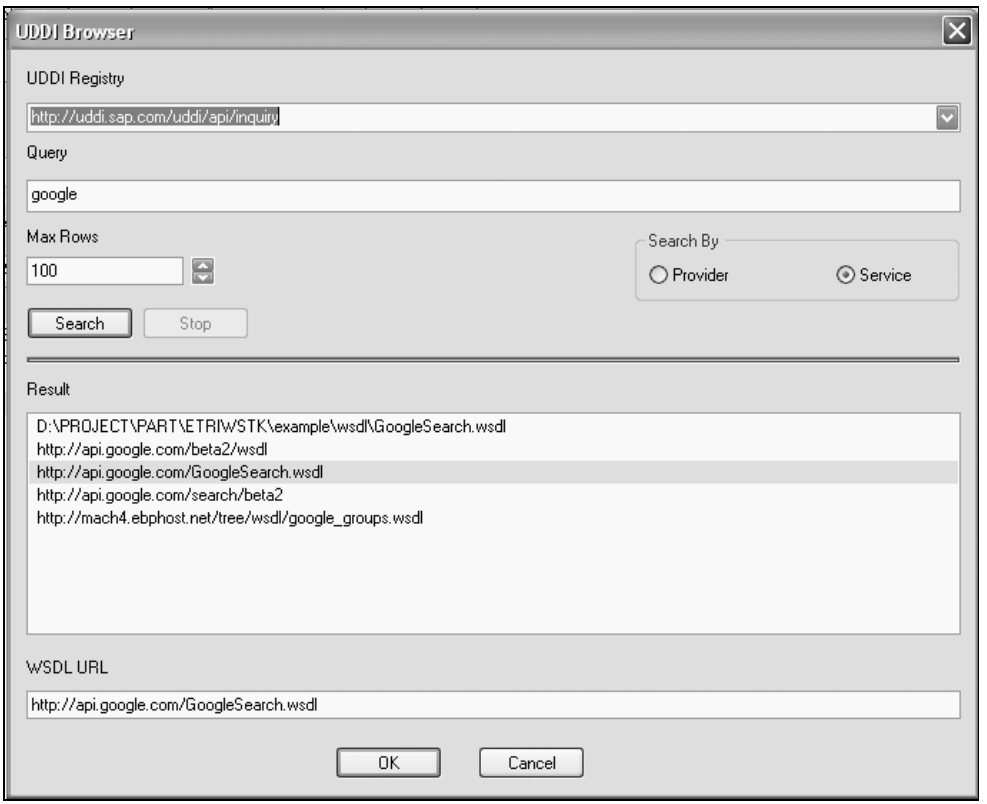


Figura 12. *Obținerea rezultatelor privitoare la documentele WSDL găsite*

Așa cum se observă și în figura 12, *Stylus* va afișa o listă cu rezultate. Pentru exemplificare, vom alege din lista de rezultate documentul WSDL cu URL-ul *http://api.google.com/GoogleSearch.wsdl*. În acest moment, avem la dispoziție documentul WSDL dorit și putem vedea ce metode pune la dispoziție serviciul Web (în acest caz, cel oferit de motorul de căutare Google).

Așa cum se observă mai jos, *Stylus* afișează operațiile oferite de serviciul Web, signatura metodelor, lista parametrilor, precum și un mesaj SOAP necesar pentru invocarea operației selectate. Interfața *Stylus* oferă posibilitatea de a modifica valoarea parametrilor, mesajul SOAP fiind automat actualizat cu noile date introduse.

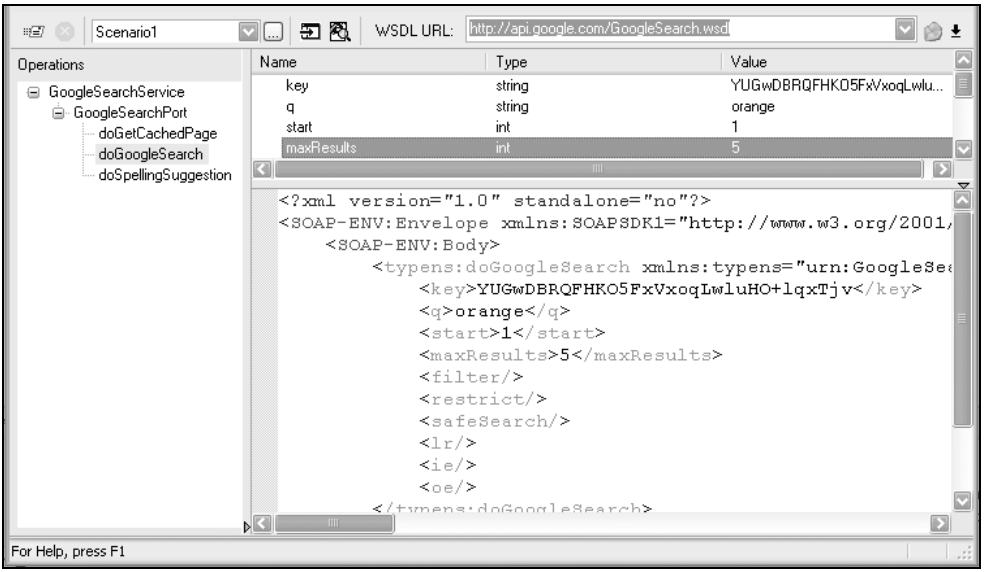


Figura 13. *Detalii privitoare la o metodă oferită de serviciul Web dorit a fi invocat*

Din acest moment suntem gata să realizăm invocarea serviciului Web (folosim butonul „*Send Request*") – a se vedea figura 14.

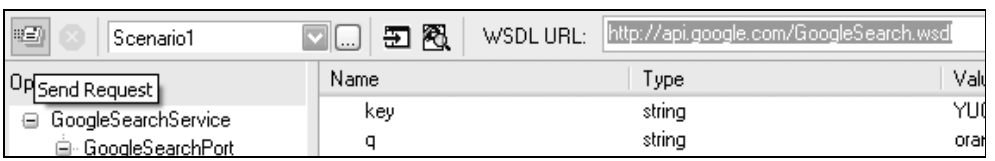


Figura 14. *Invocarea serviciului Web oferit de Google*

Stylus permite apoi inspectarea, în cadrul ferestrei *Preview*, a rezultatului întors de serverul care găzduiește serviciul Web.

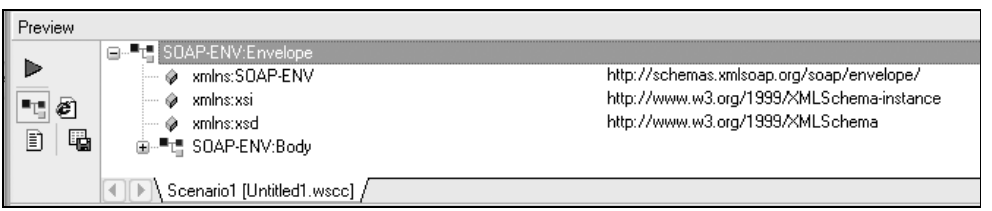


Figura 15. *Obținerea răspunsului întors de serviciul Web invocat*

Mai multe detalii despre această aplicație comercială puteți găsi la adresa *http://www.stylusstudio.com/*.

Un alt instrument este *XMLSpy* care, pe lângă alte facilități oferite, poate fi folosit pentru invocarea, editarea, vizualizarea și validarea documentelor WSDL. Să considerăm un serviciu Web de generare a numerelor prime. Pentru a-l putea invoca avem nevoie de documentul WSDL asociat, pe care îl putem găsi la adresa *http://www50.brinkster.com/vbfacileinpt/np.asmx?wsdl*.

XMLSpy permite invocarea rapidă a serviciului Web numit *Prime Numbers*, implementat în Visual Basic .NET, folosind acest document WSDL.

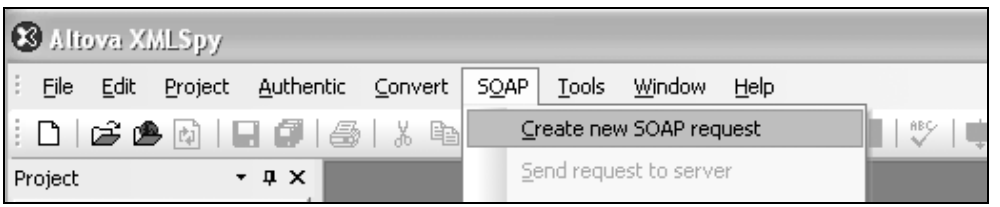

Altova XMLSpy
File Edit Project Authentic Convert SOAP Tools Window Help
Create new SOAP request
Send request to server
Project

Figura 16. *Crearea unei cereri SOAP de invocare a unui serviciu Web*

Se creează o cerere SOAP care primește ca parametru URL-ul de mai sus:

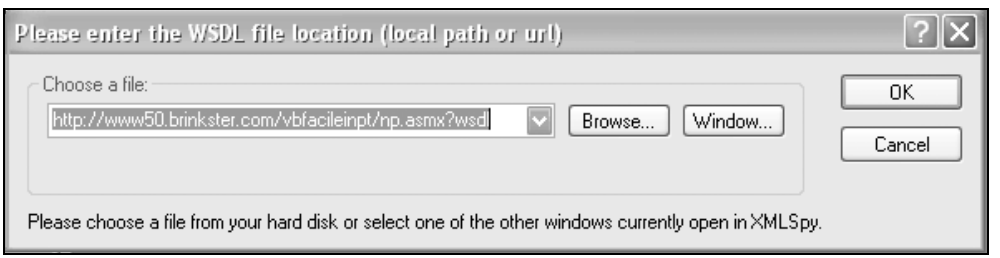

Please enter the WSDL file location (local path or url)
Choose a file:
http://www50.brinkster.com/vbfacileinpt/np.asmx?wsdl
Browse...
Window...
OK
Cancel
Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.

Figura 17. *Introducerea adresei documentului WSDL folosit pentru invocarea serviciului Web dorit*

Acest serviciu Web ne oferă o singură metodă – *GetPrimeNumbers*. Mesajul SOAP creat de instrumentul *XMLSpy* este ilustrat în figura 18.



Figura 18. *Generarea unei cereri către un serviciu Web*

Trimitem cererea SOAP și răspunsul obținut va fi:

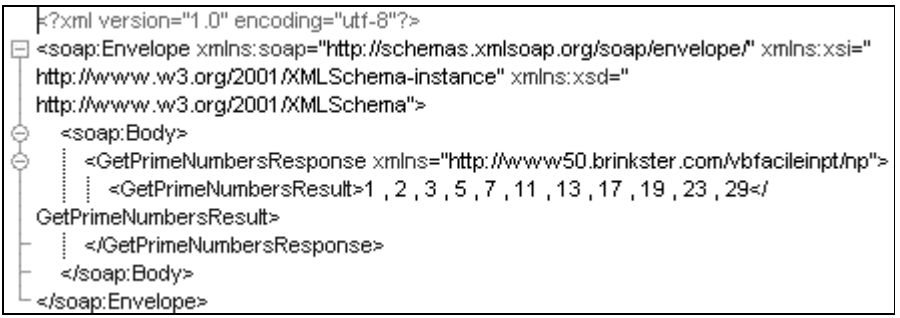


Figura 19. *Obținerea răspunsului oferit de serviciul Web invocat*

Facem observația că *XMLSpy* poate fi găsit și în cadrul mediului *Eclipse* (platformă universală de dezvoltare software implementată în Java, scopul său fiind integrarea diferitelor instrumente utile, într-o manieră indiferentă de limbajul sau platforma folosită). Această integrare permite utilizatorilor sistemului Eclipse să lucreze cu tehnologiile standard XML, utilizând numeroasele facilități oferite în acest caz de *XMLSpy*.

O altă abordare este cea oferită de *<oXygen /> XML Editor*, care pune la dispoziție un instrument de consultare și invocare via SOAP a serviciilor Web. În captura din figura 20 ilustrăm acest mecanism. Se poate observa structura fișierului WSDL returnat de Google, având acces la numele operațiilor implementate. De asemenea, *oXygen* poate fi folosit în conjuncție cu Eclipse.

Alte exemple notabile sunt *Wsdlpull* (instrument de inspectare și invocare a serviciilor Web) și *WSDL Verify* (program de verificare a corectitudinii documentelor WSDL).

În final, putem concluziona că folosirea documentelor WSDL aduce câteva avantaje majore:

- WSDL face mai facilă scrierea și menținerea serviciilor prin furnizarea unei abordări structurate, necesară definirii interfeței serviciului Web;
- specificația WSDL permite o modalitate mai eficientă de a utiliza un serviciu Web, prin faptul că reduce mărimea codului (și astfel și a potențialelor erori) pe care aplicația-client trebuie să-l implementeze;
- WSDL permite ca modificările ulterioare ale interfeței/implementării serviciului să se realizeze mai ușor. Descoperirea dinamică a descrierilor WSDL conduce la faptul că monitorizarea acestor schimbări va fi automat realizată în cazul clienților, recurgând la WSDL pentru invocarea serviciilor Web dorite.

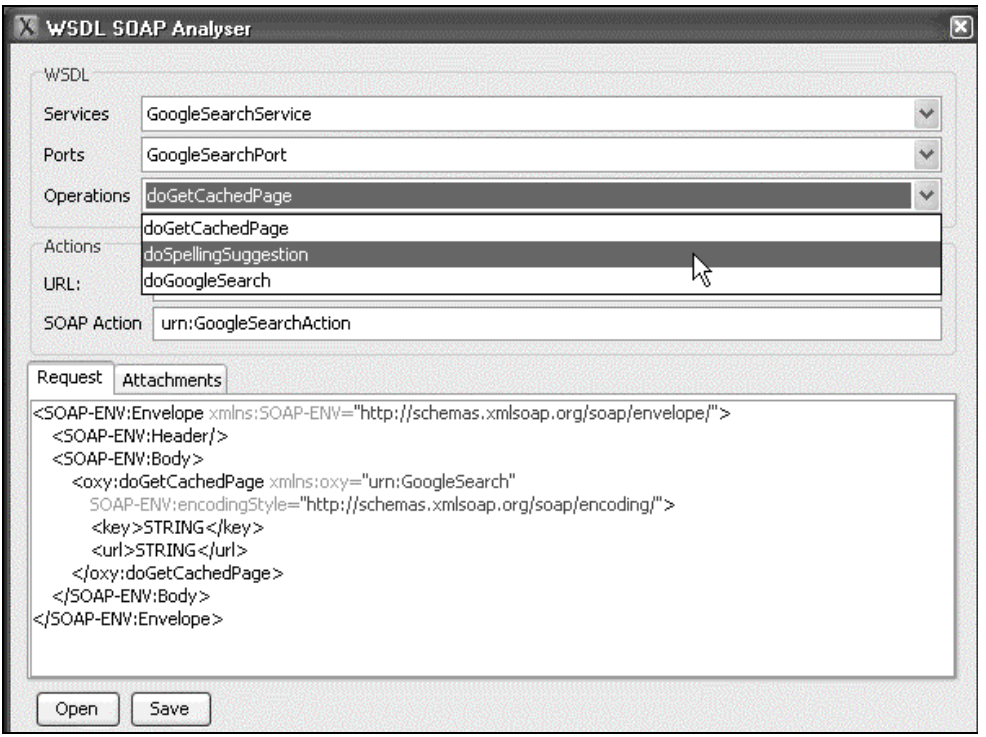


Figura 20. *Pregătirea invocării unui serviciu Web prin intermediul instrumentului* WSDL SOAP Analyser *oferit de* oXygen

Referințe

Booth, D., Liu, C. (eds.), *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, W3C Candidate Recommendation, Boston, 2006: http://www.w3.org/TR/wsdl20-primer

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006

Chinnici, R. *et al.* (eds.), *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Candidate Recommendation, Boston, 2006: http://www.w3.org/TR/wsdl20

Chinnici, R. *et al.* (eds.), *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*, W3C Candidate Recommendation, 2006: http://www.w3.org/TR/wsdl20-adjuncts

Christensen, E. *et al.*, *Web Services Description Language (WSDL) 1.1*, W3C Note, Boston, 2001: http://www.w3.org/TR/wsdl

Jorgensen, D., *Developing .NET Web Services with XML*, Syngress Publishing, 2002

Haas, H., Brown, A. (eds.), *Web Services Glossary*, W3C Working Draft, Boston, 2003: http://www.w3.org/TR/ws-gloss/

Weerawarana, S. *et al.*, *Web Services Platform Architecture*, Prentice Hall PTR, 2004
* * *, *Altova XMLSpy*: http://www.altova.com/
* * *, *Apache XML*: http://xml.apache.org/
* * *, *IBM Developer's Site*: http://www.ibm.com/developer/
* * *, *oXygen XML Editor*: http://www.oxygenxml.com/
* * *, *Portal dedicat serviciilor Web*: http://www.webservices.org/
* * *, *Stylus Studio*: http://www.stylusstudio.com/
* * *, *Wsdlpull*: http://freshmeat.net/projects/wsdlpull/
* * *, *WSDL Verify*: http://apps.gotdotnet.com/xws/wsdlverifier/wsdlverify.asmx
* * *, *XML.com*: http://xml.com/
* * *, *ZVON*: http://www.zvon.org/

Capitolul 4

Protocolul SOAP

„Totul ar trebui să fie cât mai simplu cu putință, însă nu mai simplu de atât."

(Albert Einstein)

1. Evoluția protocoalelor bazate pe XML

Tehnologia din spatele serviciilor Web este construită pe baza protocoalelor XML. Acestea guvernează modul cum are loc comunicarea între programele rulând la nivel de server și de client și cum sunt reprezentate datele transmise (parametrii de intrare și rezultatul întors).

Din punct de vedere istoric, protocoalele de comunicație bazate pe XML se împart în două generații: prima se bazează doar pe sintaxa XML 1.0, iar a două generație folosește *XML Schema* și spațiile de nume XML. SOAP se încadrează în această a doua generație.

1.1. *Prezentare succintă a protocoalelor din prima generație*

S-au făcut numeroase eforturi pentru a crea protocoale bazate pe XML. Două dintre ele s-au impus în mod deosebit: WDDX (*Web Distributed Data Exchange*) și XML-RPC.

WDDX a fost creat în 1998 de Allaire Corporation (preluată de Macromedia, devenită în acest moment parte a companiei Adobe). WDDX oferea un mecanism independent de limbaj și platformă pentru schimbul de date între aplicații. Deși WDDX suporta tipuri de date precum șiruri de caractere, numere, valori booleene, tablouri și structuri, nu putea fi folosit pentru reprezentarea oricăror tipuri de date ce pot fi exprimate în XML.

WDDX nu era legat în mod strict de o modalitate de transport, astfel că aplicațiile puteau interschimba pachete WDDX folosind diverse protocoale precum HTTP sau SMTP. În trecut, WDDX s-a bucurat de un anume succes, existând implementări în majoritatea limbajelor de programare.

XML-RPC este un protocol bazat pe paradigma RPC (*Remote Procedure Call*), descrisă în primul capitol al acestui volum. XML-RPC a fost introdus în 1998 de compania Userland. Tipurile de date suportate sunt similare cu cele din WDDX. Ca protocol de transport se folosește HTTP.

Un mesaj XML-RPC reprezintă de fapt o cerere HTTP prin metoda POST. Corpul cererii are forma unui document XML. Procedura se execută pe server, iar rezultatul este întors, de asemenea, sub formă de XML.

Parametrii de intrare ai procedurii invocate pe server pot fi numere, șiruri de caractere etc., sau pot fi structuri ori liste de structuri.

Un exemplu de cerere XML-RPC este următorul, în care se dorește invocarea funcției *furnizeazaNumeSortiment(int)*, care va oferi numele unui sortiment de portocale pe baza unui index (în acest caz, indexul are valoarea 2):

```
POST /RPC2 HTTP/1.0
User-Agent: Test/3.2 (WinXP)
Host: tsxmlrpc.ex.ro
Content-Type: text/xml
Content-Length: 181

<?xml version="1.0"?>
<methodCall>
   <methodName>furnizeazaNumeSortiment</methodName>
   <params>
      <param>
         <value><int>2</int></value>
      </param>
   </params>
</methodCall>
```

Observăm că în antetul cererii URI-ul nu este specificat. De exemplu, el ar putea fi vid (doar un singur „/”), în cazul în care serverul manipulează doar apeluri XML-RPC.

Oricum, dacă serverul operează cereri HTTP de diferite tipuri, atunci pe baza valorii URI mesajele se dirijează spre codul responsabil cu cererile XML-RPC. În exemplul de mai sus, URI-ul este */RPC2*, aceasta indicând serverului dirijarea cererilor spre aplicația „*RPC2*”, care va trimite răspunsul. Este obligatoriu ca în antet să fie specificate câmpurile *User-Agent* și *Host*.

Valoarea câmpului *Content-Type* este *text/xml*, deoarece transmitem un document XML, iar *Content-Length* reprezintă dimensiunea care trebuie corect specificată.

Corpul mesajului este marcat în XML și conține, în acest exemplu, o singură structură <*methodCall*>. Elementul <*methodCall*> trebuie să includă un

sub-element <*methodName*>, care indică (sub forma unui șir de caractere) numele metodei apelate.

Dacă procedura invocată la distanță necesită parametri, atunci <*methodCall*> trebuie să conțină și elementul-copil <*params*>. Acesta va include un anumit număr de elemente <*param*>, fiecare având o valoare de un anumit tip. În exemplul anterior, tipul parametrului este *int* și este furnizat chiar de marcatorul <*int*>.

Un exemplu de răspuns pentru cererea de mai sus va avea forma:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Mon, 20 Feb 2006 19:33:07 GMT
Server:Test/3.2-WinXP

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value>
            <string>Portocale japoneze</string>
         </value>
      </param>
   </params>
</methodResponse>
```

Dacă nu apar erori, se întoarce codul de stare *200 OK*. Valoarea câmpului de antet *Content-Type* este *text/xml*, deoarece răspunsul este și el un document XML. Câmpul *Content-Length* trebuie să fie prezent și să aibă o valoare corectă.

Corpul mesajului este o structură XML <*methodResponse*>, care conține un singur parametru <*params*> cu un unic element-copil <*param*>. Acesta, la rândul lui, include o singură valoare <*value*>.

Elementul <*methodResponse*> putea conține, de asemenea, marcatorul <*fault*> având sub-elementul <*value*>, ce reprezintă un <*struct*> compus din două elemente: primul este <*faultCode*>, de tip <*int*>, iar celălalt este numit <*faultString*> și este de tip <*string*>. Elementul <*fault*> indică apariția unei excepții XML-RPC, serverul întorcând codul acesteia și un mesaj explicativ.

De notat că <*methodResponse*> poate conține fie <*fault*>, fie <*params*>, dar nu pe ambele simultan.

Pentru cererea anterioară putem avea și următoarea variantă de răspuns, indicând o excepție referitoare la furnizarea unui număr prea mare de parametri de intrare:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Mon, 20 Feb 2006 19:33:09 GMT
Server: Test/3.2-WinXP

<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>4</int></value>
            </member>
            <member>
               <name>faultString</name>
               <value>
                  <string>Too many parameters.</string>
               </value>
            </member>
         </struct>
      </value>
   </fault>
</methodResponse>
```

Datorită simplității sale, XML a fost acceptat pe scară largă. Astfel, XML-RPC poate fi folosit în cadrul programelor Perl, Java, Python, C, C++, PHP și multe alte limbaje. Implementările sunt disponibile pentru majoritatea sistemelor de operare. Un exemplu de aplicație Web scrisă în PHP și utilizând XML-RPC este oferit în cadrul lucrării lui Sabin Buraga (coord.), *Aplicații Web la cheie*, Polirom, Iași, 2003.

1.2. *Dezavantajele protocoalelor din prima generație*

Enumerăm în cele ce urmează principalele neajunsuri ale primei generații de protocoale de transport bazate pe XML:

Lipsa de extensibilitate. Creatorii protocolului trebuie să ajungă la o înțelegere înainte ca orice modificări să fie implementate și versiunea protocolului trebuie să fie revăzută, astfel încât să se permită diferitelor instrumente să facă distincția între versiunile vechi și cele noi ale protocolului. De exemplu, atunci când în XML-RPC și WDDX s-a adăugat suport pentru datele binare, specificațiile ambelor protocoale au fost reactualizate și același lucru s-a întâmplat cu toate implementările existente pentru diferite limbaje și platforme de programare. O

astfel de necesitate constituie un serios handicap, care a fost depășit de protocoalele din a doua generație prin utilizarea spațiilor de nume XML.

A două mare problemă este legată de *tipurile de date*. Aceste protocoale foloseau un singur DTD (*Document Type Definition*) pentru a descrie reprezentarea serializată a datelor. În general se utilizau doar câteva tipuri de date simple XML. Această abordare a permis multor instrumente să ofere suport acestor protocoale. Dificultatea care a apărut ulterior a fost aceea că datele din mesaje erau exprimate sintactic și nu semantic. Protocoalele din a doua generație folosesc structurile și bogatele tipuri de date din *XML Schema*.

2. SOAP – scurtă istorie

Începând cu anul 1997, corporația Microsoft a inițiat numeroase cercetări legate de XML. Scopul era crearea de aplicații care să comunice pe baza RPC și HTTP. Companiile DevelopMentor și Userland și-au adus aportul la aceste studii. La începutul anului 1998, s-a dezvoltat conceptul de protocol de comunicații bazate pe XML, numit SOAP.

În cadrul Microsoft, grupul DCOM (*Distributed Common Object Model*) nu agrea aceasta nouă direcție și susținea că Microsoft ar trebui să folosească poziția dominantă pe care i-o oferea DCOM. Altă grupare din Microsoft era de acord cu tranziția la SOAP, însă se considera că era nevoie de ceva mai mult decât XML simplu (ar fi fost de dorit suportul oferit de specificațiile referitoare la spațiile de nume și schemele XML). Datorită acestor tergiversări, Userland a făcut public în vara anului 1998 protocolul XML-RPC.

În 1999, Microsoft a creat propria versiune de schemă XML (numită *XML Data Reduced*) și a adăugat facilități pentru suportarea spațiilor de nume. Astfel, protocolul SOAP începea să capete contur. Era totuși un mecanism de transport XML bazat pe RPC. În aceeași perioadă, o echipă care lucra la serverul BizTalk a creat un model care se focaliza mai puțin pe RPC și mai mult pe ideea de transfer de mesaje (*messaging*).

Pentru a rezolva aceste diferențe au fost necesare câteva luni. În septembrie 1999, apare versiunea publică SOAP 0.9, dată spre aprobare la IETF (*Internet Engineering Task Force*). După câteva schimbări, în decembrie 1999 e disponibilă varianta SOAP 1.0.

Pe 8 mai 2000, SOAP versiunea 1.1 a fost prezentată ca document al Consorțiului Web, la care IBM era co-autor al protocolului. Această specificație se caracteriza printr-o mai mare extensibilitate și a eliminat îngrijorările conform cărora implicarea Microsoft ar conduce dezvoltatorii la necesitatea utilizării unor tehnologii proprietare.

În curând, după apariția specificației SOAP 1.1, IBM a lansat pe piață implementarea Java SOAP, pe care a cedat-o mai apoi proiectului Apache XML (*http://xml.apache.org/*), în scopul unei dezvoltări *open source*. Ulterior, mai multe corporații au început să lucreze la implementări ale serviciilor Web folosind SOAP.

Pe 9 iulie 2001, grupul W3C a publicat o nouă schiță (*working draft*) pentru SOAP 1.2, iar la data de 7 iunie 2003, noua versiune SOAP 1.2 a devenit recomandare standardizată. Începând cu varianta 1.2, termenul „SOAP" nu se mai consideră a fi un acronim.

Ca majoritatea tehnologiilor noi, SOAP schimbă regulile după care sunt dezvoltate aplicațiile. Din experiența cu alte tehnologii, s-a observat că acelea care se caracterizează prin simplitate sunt cele mai promovate și au șansa să dureze. Așa cum se va vedea din exemplele viitoare, într-o anumită măsură, protocolul SOAP se dovedește a fi simplu de înțeles și de utilizat.

3. SOAP – o vedere de ansamblu

Serviciile Web sunt o instanță a modelului arhitecturilor orientate spre servicii (SOA – *Service Oriented Architecture*), care folosesc SOAP ca mecanism de transport a mesajelor între servicii descrise cu ajutorul unor interfețe WSDL. În esență, avem de-a face cu o arhitectură simplă (vezi figura 1) în care mesajele SOAP sunt propagate cu ajutorul unui protocol de transport între serviciul Web și posibilii săi clienți.

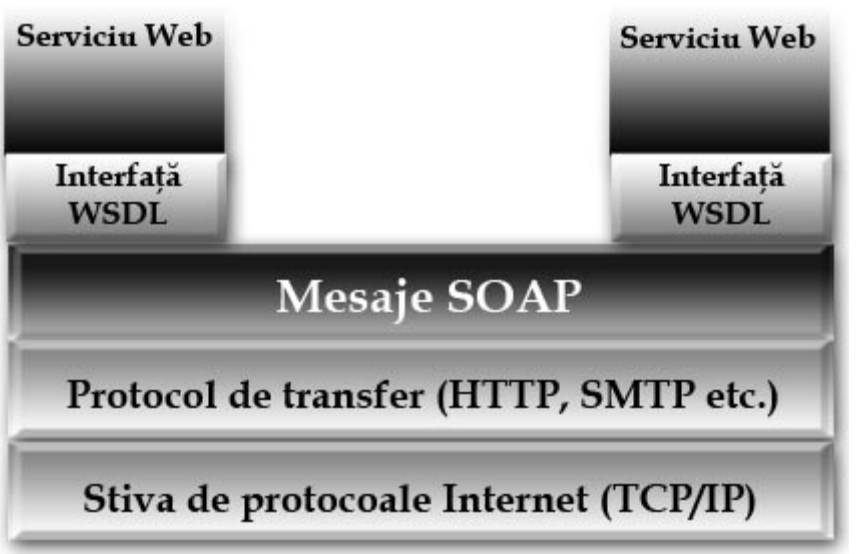


Figura 1. *Stiva logică a serviciilor Web*

SOAP reprezintă un protocol care permite atât schimbul de informații structurate într-un mediu distribuit și descentralizat, cât și accesarea de servicii, privite la nivel orientat-obiect, într-o manieră independentă de platformă.

Scopul principal al SOAP este facilitarea inter-operabilității între platforme și limbaje de programare. Pentru realizarea schimbului de informații între sisteme eterogene, părțile care comunică trebuie să cadă de acord asupra unei reprezentări comune. Așa cum se poate observa din exemplul următor, un număr de telefon poate avea mai multe reprezentări echivalente din punct de vedere logic (semantic).

```
<nrTelefon>0232 201090</nrTelefon>

<nrTelefon>
  <prefix>0232</prefix>
  <numar>201090</numar>
</nrTelefon>

<nrTelefon prefix="0232" numar="201090" />
```

Apare întrebarea: care este codificarea corectă? Răspunsul: este corectă cea pe care aplicația o așteaptă. Cu alte cuvinte, nu este suficient să afirmăm că serverul și clientul folosesc XML pentru a realiza schimbul de informații. Trebuie să definim tipul datelor interschimbate, maniera de modelare a datelor în XML, mecanismul de transmitere a informației, folosind – la nivel mai scăzut – protocoalele Internet existente.

Fără aceste convenții, programele nu pot cunoaște modul de decodificare a datelor pe care le primesc, chiar dacă sunt reprezentate cu ajutorul XML. SOAP este cel care furnizează aceste convenții.

Arhitectura generală SOAP (despre care vom discuta în detaliu în cadrul subcapitolelor următoare) este surprinsă în figura 2.

Un mesaj SOAP constă dintr-un document XML al cărui element-rădăcină are denumirea de plic (*envelope*). În interiorul mesajului întâlnim două elemente-copil cunoscute ca antet (*header*) și corp (*body*), similar mesajelor transmise prin poștă. Informațiile privind modul de interacțiune cu aplicațiile sunt plasate în interiorul corpului, iar în cadrul antetului se găsesc informații despre destinatar, despre entitatea care va avea posibilitatea modificării mesajului respectiv etc.

În acest capitol, vom prezenta o serie de exemple de mesaje SOAP sub formă de documente XML. Specificațiile SOAP se referă la un mesaj SOAP ca fiind specificat formal printr-un *XML Infoset* (vezi capitolul 2), descriere abstractă a conținutului său. Cititorii interesați de astfel de aspecte (de exemplu, asocierea SOAP cu alte protocoale, caz în care mesajele pot avea reprezentări diferite) le pot găsi în rapoartele tehnice furnizate de Consorțiul Web.

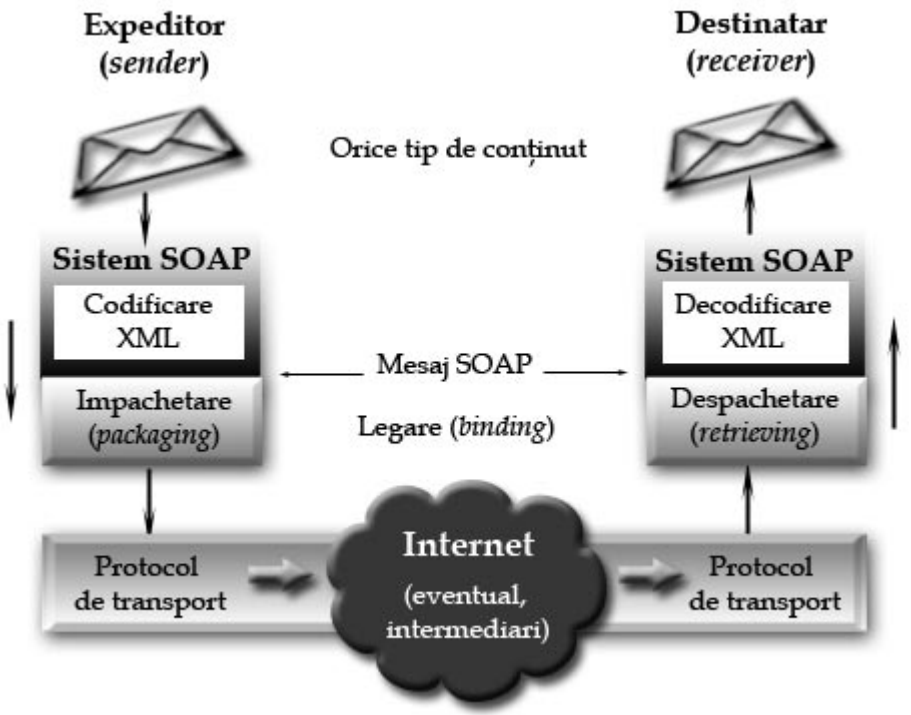


Figura 2. *Arhitectura SOAP*

4. Forma generală a unui mesaj SOAP. Procesarea datelor de către serverele SOAP

Mesajele SOAP sunt documente XML care reprezintă structura atomică pentru comunicarea în medii distribuite. Elementul-rădăcină al unui mesaj SOAP este marcatorul *Envelope*. Deoarece elementul *Envelope* este în mod unic identificat de propriul său spațiu de nume, instrumentele de procesare vor putea determina cu ușurință dacă un document XML reprezintă sau nu un mesaj SOAP.

Putem să imaginăm un mesaj SOAP ca pe un plic (*SOAP Envelope*) care conține în interior un antet (*header*) opțional și un corp (*body*) obligatoriu – a se consulta figura 3.

Antetul conține blocuri de informații relevante pentru modul cum va fi procesat mesajul. Acesta include date privitoare la locația emitentului, locația destinatarului, anumite informații de autentificare și autorizare. Corpul stochează mesajul care va fi trimis și procesat. Orice aspect ce poate fi exprimat cu ajutorul sintaxei XML poate fi memorat în corpul mesajului.

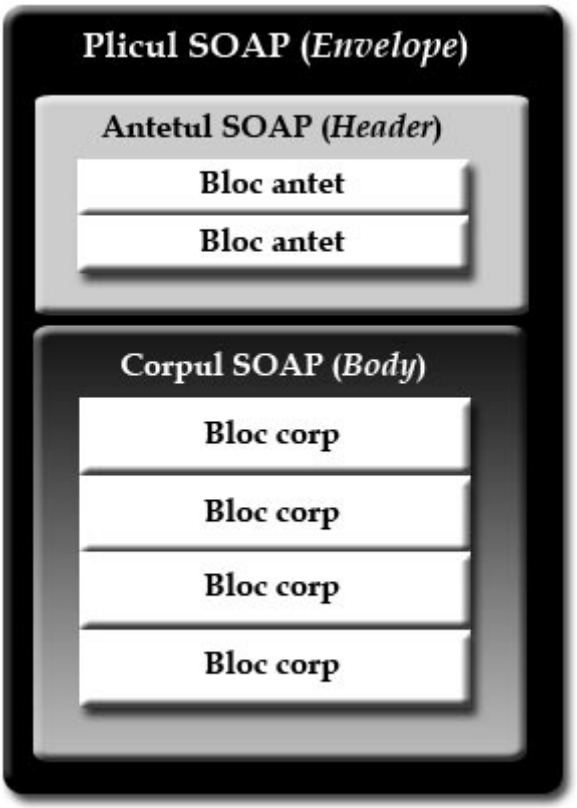


Figura 3. *Structura unui mesaj SOAP*

După cum se poate observa, există aici – ca și în cazul poștei electronice – o separare între partea destinată aplicațiilor propriu-zise și partea de transport. Informațiile destinate aplicațiilor le găsim la nivelul corpului mesajului, pe când elementele legate de transportul mesajului sunt incluse doar în partea de antet.

Ne putem imagina că o aplicație SOAP este formată din noduri (entități de calcul intermediare) care schimbă mesaje între ele. Mesajele SOAP pot trece printr-un număr oarecare de noduri intermediare până ajung la destinatarul final.

Specificațiile SOAP presupun existența unor noduri intermediare care descriu comportamentul unui nod în anumite circumstanțe. Un mesaj, care trece din nod în nod pentru a ajunge în final la destinatar, poate fi procesat de aceste noduri intermediare (vezi figura 4). Mai mult, un astfel de nod poate realiza modificări în cadrul unui mesaj, astfel încât acesta să poată fi procesat de un nod diferit de cel final.

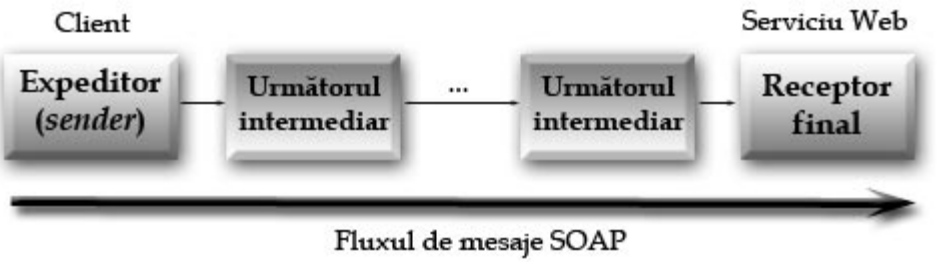


Figura 4. *Nodurile SOAP și fluxul mesajelor vehiculate*

Așa cum se observă în figura 4, nodurile intermediare primesc mesajul și îl pot altera sau nu. Nodul final se numește și *ultimate receiver*, adică este nodul terminal pe care se află serviciul Web care va folosi acel mesaj.

Întregul model de procesare este implementat de către serverele SOAP. Un server SOAP desemnează un element al mecanismului care face medierea între traficul SOAP și componentele unei aplicații. Pentru a putea dezvolta servicii Web, trebuie să înțelegem modul în care un server SOAP implementează modelul SOAP.

Pornind de la premisa că este imposibil să facem o trecere în revistă a tuturor platformelor SOAP existente, vom prezenta arhitectura generală a unui server SOAP. De asemenea, vom enumera principalele caracteristici de care se bucură cele mai populare implementări actuale – a se vedea și capitolul 6 pentru exemple de aplicații (servere și clienți) utilizând SOAP.

O imagine generală a unui server SOAP poate fi urmărită în figura 5.

Considerăm scenariul în care un mesaj SOAP a ajuns la un server SOAP.

Așa cum se poate observa, elementele stocate în antetul mesajului SOAP (de exemplu, informații caracteristice pentru alte servicii Web cum ar fi securitatea, tranzacțiile etc.) sunt procesate de anumiți „lucrători" (*handlers*) care au fost înregistrați pentru serviciul Web în cauză. Astfel de entități (în fapt, funcții de tratare a antetelor) sunt considerate – în termenii SOAP – noduri intermediare.

Rutinele de tratare a antetelor nu fac implicit parte din serverele SOAP, dar sunt în general utilizate pentru îmbunătățirea capacităților serverului. Aceasta înseamnă că serverele SOAP pot fi la rândul lor perfecționate ulterior, pentru a fi compatibile cu noile tehnologii/servicii Web care apar.

Dacă în urma procesării antetului nu s-a obținut nici o eroare, partea de mesaj care e destinată aplicației (care se găsește în corpul SOAP) ajunge la nivelul de dispecer (*dispatch*), care transmite implementării serviciului datele propriu-zise de intrare. După efectuarea procesării în interiorul serviciului, are loc trimiterea răspunsului către *dispatcher*, care-l va propaga către entitatea ce a formulat cererea. Așa cum se poate remarca din figura 5, mesajul de răspuns poate trece

prin mai multe etape de transformări. Anumiți *handler*-i (noduri intermediare) pot modifica antetul mesajului. Altfel spus, dacă revenim la analogia mesaj SOAP – plic, e ușor să ne imaginăm cum acesta ajunge să fie adus de un poștaș la destinație, dar înainte de aceasta a trecut prin mai mulți „intermediari" (societăți de transport, oficii poștale etc.) care l-ar fi putut modifica. Aceste modificări sunt realizate doar la exteriorul plicului (de exemplu, diverse ștampilări), conținutul lui rămânând intact până ajunge la destinatar.

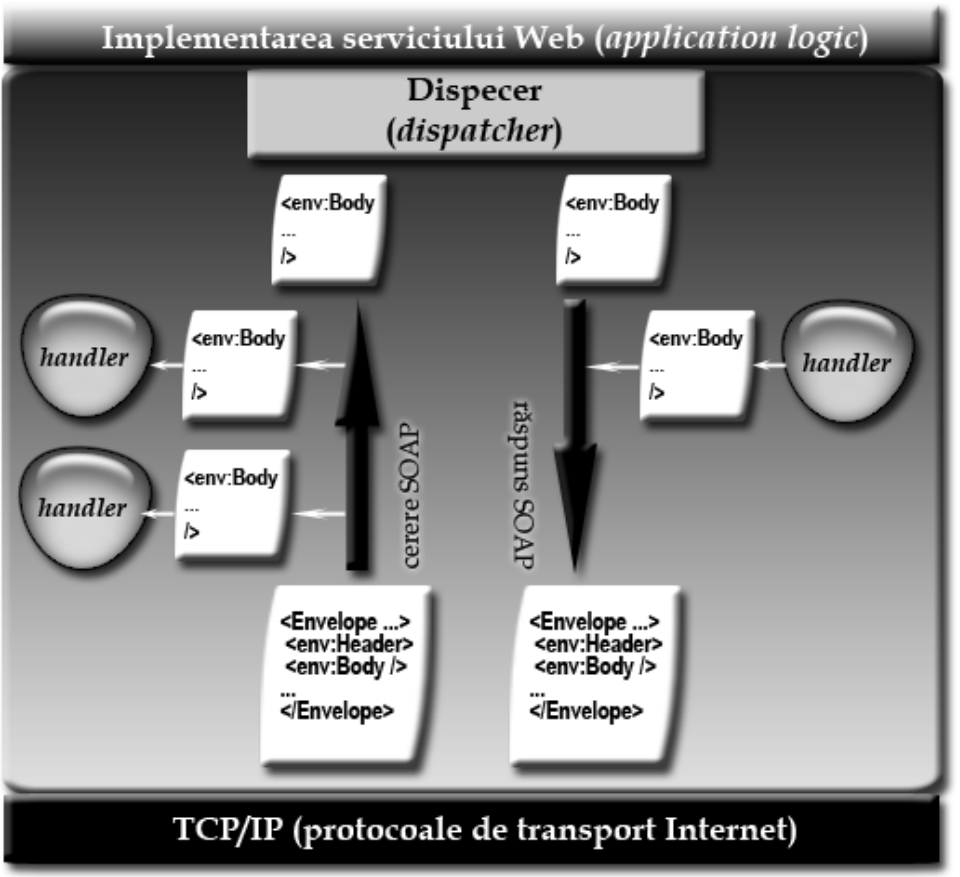


Figura 5. *Arhitectura generală a unui server SOAP*

Eventual, mesajul poate ajunge la un nivel de rețea în care este transformat (*marshalled*) pentru a suporta un anumit protocol de nivel scăzut (de exemplu, SMTP ori HTTP), însă va fi trimis într-un mod corespunzător nodurilor SOAP pentru a putea fi procesat.

5. Modelul SOAP – mesaje și codificări

Pornind de la premisa că toate cele discutate până acum v-au stârnit curiozitatea și v-au familiarizat cu noțiunile importante ale protocolului, vom încerca să prezentăm în detaliu elementele care contribuie la crearea modelului SOAP.

Specificația SOAP este divizată în trei componente:

– prima constă într-o descriere a utilizării protocolului SOAP, a structurii mesajelor și a modelului conform căruia se realizează schimbul de mesaje SOAP;
– partea a doua (*Messaging Framework*) furnizează un model în care se specifică regulile de procesare a unui mesaj SOAP, un cadru de lucru extensibil care permite dezvoltatorilor să utilizeze extensii în cadrul sau în afara mesajului SOAP, reguli pentru construcția mesajelor SOAP și legături între SOAP și diferite protocoale (cum ar fi HTTP);
– partea a treia (*Adjuncts*) prezintă, printre altele, regulile pentru realizarea unui apel SOAP RPC și regulile de codificare a mesajelor SOAP.

În secțiunile următoare, vom prezenta modelul de bază și structura documentelor XML, schemele de codificare, convențiile RPC utilizate și legătura SOAP cu protocoalele de transport.

5.1. *Mesajele SOAP*

Dacă până în acest moment ne-am făcut o idee generală privitoare la structura per ansamblu a unui mesaj SOAP, în această secțiune vom începe studierea concretă a unor exemple.

Reamintim că un mesaj SOAP trebuie să includă cel puțin un bloc de date în corpul mesajului și că antetul este opțional. De altfel, nu există o limită superioară a numărului de blocuri ce pot apărea în antet sau în corp.

Fie următorul exemplu de mesaj SOAP, vehiculând date referitoare la achizițiile și vânzările unor sortimente de portocale:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope">
  <env:Header>
  <tz:tranzactie-id
         xmlns:tz="http://tranzactie.portocale.info"
         env:encodingStyle=
```

```
            "http://tranzactie.portocale.info/enc"
          env:role=
            "http://www.w3.org/2002/06/soap-
             envelope/role/ultimateReceiver"
          env:mustUnderstand="true">
        tranz10-01
   </tz:tranzactie-id>
   </env:Header>
   <env:Body xmlns:p="http://www.portocale.info/">
     <p:portocale>
       <!-- partea de achiziţie de portocale -->
       <p:achizitii env:encodingStyle=
          "http://www.w3.org/2002/06/soap-encoding">
          <p:tip>Portocale greceşti fără coajă</p:tip>
          <p:cod>P10-01-GR</p:cod>
          <p:cantit p:um="kg">3374</p:cantit>
       </p:achizitii>
       <!-- partea de vânzări de portocale -->
       <p:vanzari>
          <p:tip>Portocale japoneze albastre</p:tip>
          <p:cod>P10-03-JP</p:cod>
          <p:cantit p:um="kg">0107</p:cantit>
       </p:vanzari>
     </p:portocale>
   </env:Body>
</env:Envelope>
```

Dacă urmărim structura acestui mesaj, observăm că respectă modelul prezentat în figura 2. Mesajul din exemplul nostru conţine un singur bloc-antet, iar în corp întâlnim două elemente (care vor fi utilizate de destinatarul final). Atât elementul *Header*, cât şi *Body* sunt copiii marcatorului *Envelope* care, de fapt, joacă rolul unui container de date.

5.2. *Elementul* Envelope

Elementul *Envelope* este elementul în interiorul căruia se găseşte mesajul SOAP. Acest element are asociat spaţiul de nume de la adresa *http://www.w3.org/2002/06/soap-envelope*.

În exemplul anterior, identificarea acestui spaţiu de nume se face prin prefixul *env*.

```
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope">
   <!-- parte opţională -->
```

```
  <env:Header>
  ...
  </env:Header>
  <!-- un singur element Body (obligatoriu) -->
  <env:Body xmlns:p="http://www.portocale.info/">
  ...
  </env:Body>
</env:Envelope>
```

Elementul *Envelope* poate avea maxim două elemente-copil: *Header* și *Body*. De asemenea, *Envelope* trebuie să conțină și declarațiile spațiilor de nume care sunt folosite în interiorul mesajului.

5.3. *Elementul* Header

Dacă acest element apare, atunci el trebuie să fie primul element-copil al marcatorului SOAP *Envelope*. Elementul *Header* poate conține un set de intrări, fiecare element încapsulat de *Header* purtând numele de *bloc antet*.

Scopul unui bloc antet este de a comunica informația relevantă în procesarea mesajului SOAP. Un exemplu ar putea fi un bloc antet care conține elemente de autentificare sau informații despre ruta (drumul) mesajului.

Elementul *Header* are asociat spațiul de nume specificat de adresa *http://www.w3.org/2002/06/soap-envelope*.

```
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope">
  <env:Header>
  <tz:tranzactie-id
         xmlns:tz="http://tranzactie.portocale.info"
         env:encodingStyle=
           "http://tranzactie.portocale.info/enc"
         env:role=
           "http://www.w3.org/2002/06/soap-
            envelope/role/ultimateReceiver"
         env:mustUnderstand="true">
       tranz10-01
  </tz:tranzactie-id>
  </env:Header>
  ...
</env:Envelope>
```

Fiecare bloc antet trebuie să aibă asociat un spațiu de nume calificat (în acord cu regulile din *SOAP Schema*). Modul de codificare a datelor se realizează prin intermediul atributului *encodingStyle*, iar nodul care trebuie să-l proceseze se specifică

prin atributul *role*. De asemenea, poate conține și atributul *mustUnderstand*, cu ajutorul căruia se poate solicita ca un nod SOAP să-l înțeleagă sau nu.

În specificațiile SOAP este clar definit faptul că atributele *role* și *mustUnderstand* pot apărea doar în declarațiile blocurilor antet. De altfel, emițătorul unui mesaj SOAP trebuie să plaseze aceste atribute cu mare grijă. Dacă ele sunt amplasate greșit, atunci receptorul poate să le ignore. Deoarece aceste atribute sunt fundamentale pentru procesarea SOAP, le vom detalia în cele ce urmează.

Atributul role

Atributul *role* controlează nodul SOAP (îl vom numi nod-țintă) care va putea procesa unul sau mai multe blocuri antet. Atributul *role* conține un URI care identifică rolul pe care îl va juca nodul-țintă. Atunci când un nod primește un mesaj incluzând blocuri antet, el trebuie să verifice aceste blocuri pentru a determina dacă există unul pe care-l poate prelucra.

Deși unui atribut *role* i se poate asocia orice URI, specificația SOAP a prevăzut trei situații standard:

– *http://www.w3.org/2002/06/soap-envelope/role/none* – în acest caz, nici un nod SOAP nu ar trebui să încerce să proceseze respectivul bloc antet, deși alte blocuri antet pot conține referințe la el și la conținutul său;
– *http://www.w3.org/2002/06/soap-envelope/role/next* – în această circumstanță, fiecare nod trebuie să recunoască faptul că respectivul bloc antet trebuie trimis intact următorului nod SOAP (din cadrul rutei de transport a mesajului);
– *http://www.w3.org/2002/06/soap-envelope/role/ultimateReceiver* – în acest caz, blocurile antet ce conțin atributul *role* cu acest URI asociat (sau cele care nu au atributul *role*) vor fi procesate doar de către destinatarul final.

În exemplul dat mai sus, atributul *role* are ca valoare URI-ul *http://www.w3.org/2002/06/soap-envelope/role/ultimateReceiver*, ceea ce înseamnă că destinatarul ultim este cel care va putea procesa acest antet. Acest ultim nod (pe care se găsește măcar un serviciu Web), trebuie să fie capabil să proceseze conținutul mesajului (inclus în elementul *Body*).

Atributul mustUnderstand

Dacă atributul *mustUnderstand* are valoarea *true*, atunci nodul care primește mesajul trebuie să-l poată procesa sau să emită un mesaj de eroare corespunzător în caz contrar. Blocurile antet care conțin `mustUnderstand="true"` sunt cunoscute sub numele de blocuri antet obligatorii. Anteturile care nu au specificat atributul *mustUnderstand* sunt totuși examinate de nodurile-țintă.

În exemplul anterior atributul *mustUnderstand* are valoarea *true*, aceasta însemnând că destinatarul final este obligat să proceseze antetul. Altfel, nu poate să prelucreze corpul mesajului.

În specificațiile SOAP se precizează că un emițător SOAP nu trebuie să genereze un mesaj care să aibă atributul *mustUnderstand* cu valoarea 1 sau 0, ci *true*, respectiv, *false*. Totuși, nodurile SOAP care primesc mesajul trebuie să accepte și mesajele pentru care atributul *mustUnderstand* are valoarea 0 sau 1.

Atributul encodingStyle

Atributul *encodingStyle* este folosit pentru a declara modul prin care conținutul unui bloc antet a fost creat. Această informație permite nodurilor care interacționează cu acel mesaj să poată decodifica informațiile pe care le conține. SOAP permite mai multe scheme de codificare, dar furnizează și o schemă proprie.

Deoarece acest atribut este utilizat și în corpul mesajului, vom discuta mai multe despre el în cadrul secțiunii 5.4.

Atributul relay

În SOAP 1.2, pentru blocurile antet este definit și atributul opțional *relay*, cu valori de tip boolean. Acest atribut specifică dacă un bloc antet poate fi transmis mai departe, în cazul în care nodul intermediar care trebuia să-l proceseze nu a realizat acest lucru. Există precizată regula conform căreia un bloc antet care a fost deja prelucrat trebuie șters.

Sunt situații în care proiectantul unui aplicații ar dori să introducă noi facilități în aplicație cu ajutorul unor blocuri antet. Un astfel de bloc antet va putea fi procesat de nodurile intermediare care îl „înțeleg" și va fi ignorat de celelalte. În mod evident, facilitatea nou apărută nu a fost implementată de toate nodurile SOAP. Astfel, pentru a evita situațiile în care un nod SOAP nu recunoaște acel bloc antet și generează o eroare, se va preciza `env:mustUnderstand="false"`. Suplimentar, pentru a împiedica aplicarea regulii implicite a mecanismului de procesare, blocul antet va avea asociat și atributul `env:relay= "true"` (blocul nu va fi eliminat).

Exemplul următor prezintă utilizarea atributului *relay:*

```
<?xml version="1.0" ?>
<env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
      <a:firstQuestion
         xmlns:a="http://www.intrebare.ro"
         env:role="http://www.intrebare.ro/Log"
```

```
          env:mustUnderstand="true">
      ...
      </a:firstQuestion>
      <b:secondQuestion
        xmlns:b="http://www.intrebare.ro"
        env:role="http://www.w3.org/2003/05/
                  soap-envelope/role/next"
        env:relay="true">
      ...
      </b:secondQuestion>
      <c:thirdQuestion
         xmlns:c="http://www.intrebare.ro">
      ...
      </c:thirdQuestion>
   </env:Header>
   <env:Body>...</env:Body>
</env:Envelope>
```

Observăm în exemplul dat că blocul antet *b:secondQuestion* este țintă pentru următorul (*next*) nod din calea de mesaje și are prevăzut și atributul `relay="true"`. Un nod SOAP care primește un asemenea mesaj poate procesa acest bloc antet în cazul în care îl poate înțelege, dar realizând aceasta conform regulilor existente, el va trebui să șteargă acest bloc antet înainte de a retransmite mesajul. Oricum, dacă nodul SOAP ignoră acest bloc antet (și poate efectua acest lucru, deoarece blocul nu trebuie obligatoriu să fie procesat), atunci el va trebui să-l retrimită.

Procesarea blocului antet *a:firstQuestion* este obligatorie. În acest caz, blocul trebuie șters înainte ca mesajul să fie retransmis. Există și cazul de excepție când nodul, procesând și alte blocuri antet, găsește reguli care impun ca acel bloc să nu fie eliminat.

Blocul antet *c:thirdQuestion* nu are prevăzut atributul *relay*, această situație fiind echivalentă cu `relay="false"`. Conform regulilor, acest bloc antet nu este trimis mai departe dacă nu este procesat.

Blocul antet Representation

Blocul antet *Representation* permite mesajelor SOAP să transporte reprezentări ale resurselor Web.

Utilizarea unui astfel de bloc se face în cazul în care receptorul are o capacitate limitată de a obține reprezentări prin alte mijloace, de exemplu din cauza restricțiilor de accesare sau pentru că efortul de calcul ar fi prea mare (situație întâlnită, de pildă, pentru anumite dispozitive mobile). Acest bloc este, de asemenea, folositor atunci când sunt necesare mai multe referințe la aceeași resursă, dar duplicarea resurselor nu este de dorit.

În același mesaj SOAP pot exista mai multe apariții ale blocului *Representation*. Să urmărim următorul exemplu:.

```
<soap:Envelope
  xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
  xmlns:rep='http://www.w3.org/2004/08/representation'
  xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Header>
    <rep:Representation resource=
      'http://www.infoiasi.ro/~adria/foto.jpg'>
      <rep:Data xmlmime:contentType=
         'image/jpg'>/aWKKapGGyQ=</rep:Data>
    </rep:Representation>
  </soap:Header>
  <soap:Body>
    <a:Persoana
      xmlns:a='http://www.infoiasi.ro/~adria/mystuff'>
      <a:nume>Alboaie Lenuța</a:nume>
      <a:foto
         uri='http://www.infoiasi.ro/~adria/me.jpg' />
    </a:Persoana>
  </soap:Body>
</soap:Envelope>
```

Elementul *Representation* poate avea atributele *resource* și *reinsert*, conținând obligatoriu marcatorul *Data*. Valoarea atributului *resource* identifică resursa Web a cărei reprezentare este transportată de elementul *Representation*. Atributul *reinsert* este de tip boolean. Când are valoarea *true* înseamnă că nodul intermediar care procesează mesajul trebuie să reinsereze blocul antet. Aceasta înseamnă că, în conjuncție cu atributul *relay* cu valoarea *true*, blocul antet *Representation* trebuie retrimis. Dacă acest atribut nu există sau valoarea lui este *false*, atunci se aplică regulile obișnuite de procesare SOAP.

Valoarea elementului *Data* este o codificare în baza 64 (*base64*) a reprezentării unei resurse Web transportată de elementul *Representation*.

5.4. *Corpul mesajului SOAP. Elementul* Body

Elementul *Body* este un simplu container pentru datele XML destinate aplicației. Dacă pentru *Header* trebuie respectate o serie de reguli, pentru *Body* nu există o formă standard a structurii sau o modalitate strictă de interpretare. Singura constrângere este ca destinatarul final (*ultimateRecipient*) să poată procesa conținutul mesajului.

Dacă privim cu atenție elementele *Header* și *Body* observăm că, deși sunt definite ca elemente independente, ele sunt totuși interconectate. Relația dintre un bloc *Body* și unul *Header* este următoarea: o intrare *Body* este din punct de vedere semantic echivalentă cu una *Header* care nu are specificat un atribut *role* și pentru care atributul *mustUnderstand* prezintă valoarea *true*.

Deja am furnizat exemple de conținuturi stocate de marcatorul *Body*, astfel încât putem să ne referim acum la semnalarea excepțiilor de comunicare.

Coduri de eroare SOAP

În afară de rolul său de a conține datele destinate prelucrării de către o aplicație, *Body* este folosit și pentru trimiterea (către client, uzual) de excepții care pot apărea în comunicarea de date. Astfel, este utilizat elementul *Fault* pentru transportul informațiilor (structurate sau nestructurate) legate de problemele care pot apărea – de exemplu, în momentul procesării unui mesaj. Acest mecanism de management al erorilor este prevăzut de specificațiile SOAP, de aceea instrumentele SOAP trebuie să respecte acest standard pentru tratarea erorilor.

Elementul *Fault* are același spațiu de nume ca *Envelope* și conține obligatoriu două elemente-copil: *Code* și *Reason*. De asemenea, poate include opțional și elementele *Node*, *Role* și *Detail*.

Un exemplu de utilizare a elementului *Fault* îl avem în fragmentul de cod de mai jos. Acest mesaj de eroare este un răspuns la mesajul din exemplul privitor la tranzacțiile de sortimente de portocale (a se revedea secțiunea 5.1). Pentru a înțelege ce a cauzat eroarea trebuie să cunoaștem semnificația fiecărui element prezent în mesaj.

```
<?xml version="1.0" ?>
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope"
   xmlns:p="http://www.portocale.info/">
   <env:Body>
      <env:Fault>
         <env:Code>
            <env:Value>env:Receiver</env:Value>
            <env:Subcode>
               <env:Value>p:cod_incorect</env:Value>
            </env:Subcode>
         </env:Code>
         <env:Reason lang="en-UK">
           The specified product does not exist.
         </env:Reason>
```

```
            <env:Detail>
               <err:DetaliiEroare xmlns:err=
                  "http://www.portocale.info/fault">
                  <err:codAchizitInvalid>
                     <p:cod>P10-01-GR</p:cod>
                  </err:codAchizitInvalid>
               </err:DetaliiEroare>
            </env:Detail>
         </env:Fault>
      </env:Body>
   </env:Envelope>
```

Urmărind codul exemplului, remarcăm faptul că primul element al lui *Fault* este *Code*, incluzând două sub-elemente: primul este obligatoriu și se numește *Value*, iar al doilea e opțional și are denumirea de *Subcode*.

Elementul *Value* poate conține orice cod de eroare care se găsește în spațiul de nume specificat de *http://www.w3.org/2002/06/soap-envelope*.

În cazul nostru, conținutul marcatorului *Value* este *env:Receiver* (vezi exemplul următor) și ne indică faptul că a existat un nod la capătul rutei de mesaje (adică destinatarul final) care a generat eroarea. Astfel, putem ști că această eroare nu a fost generată de un nod intermediar care a procesat antetul mesajului:

```
<env:Code>
  <env:Value>env:Receiver</env:Value>
</env:Code>
```

Elementul *Subcode* include elementul *Value*, care oferă informații despre erorile specifice aplicației respective (se folosește spațiul de nume definit de dezvoltatorul acelei aplicații).

```
<env:Subcode>
  <env:Value>p:cod_incorect</env:Value>
</env:Subcode>
```

Tabelul următor enumără codurile de eroare definite de specificația SOAP.

Cod de eroare	Descriere
Version Mismatch	Apare când se sesizează o diferenţă între versiunile SOAP folosite.
MustUnderstand	Apare dacă blocul antet conţinând atributul *mustUnderstand="true"* nu a putut fi procesat de receptor.

Cod de eroare	Descriere
DataEncodingUnknown	Apare atunci când conţinutul fiecărui bloc antet sau bloc din corpul mesajului este codificat cu ajutorul unei scheme despre care nodul SOAP raportează că nu o poate înţelege.
Sender	Apare atunci când emiţătorul trimite un mesaj care nu are un format corect (de exemplu, un mesaj care nu conţine suficiente date astfel încât destinatarul să-l poată procesa).
Receiver	Apare atunci când entitatea ce recepţionează mesajul nu-l poate procesa din cauza unor erori ale aplicaţiei. Presupunându-se că eroarea nu este persistentă, mesajul se poate retransmite.
Server	Clasa de erori *Server* precizează că mesajul nu poate fi prelucrat, nu din motive legate strict de conţinutul mesajului, ci mai degrabă din cauza unor probleme de procesare. De exemplu, serviciul ar putea include o conexiune la un terţ server care nu răspunde. Mesajul ar putea fi totuşi procesat cu succes într-un alt moment de timp.

Deşi nu-l găsim în exemplul nostru, elementul *Subcode* determină ca mecanismul de tratare a erorilor în SOAP să fie extensibil. Astfel, *Code* conţine un sub-element obligatoriu *Value* şi unul opţional *Subcode*, care poate la rândul lui să includă alte marcaje *Subcode*.

Elementul *Reason* asociat lui *Code* este utilizat pentru a furniza o explicaţie în limbaj natural. În exemplul nostru, avem un mesaj emis în limba engleză: „*The specified product does not exist*" („Produsul specificat nu există"). Instrumentele SOAP folosesc adesea conţinutul elementului *Reason* atunci când semnalează erori, astfel încât activitatea de depanare să decurgă mai uşor.

Elementul opţional *Node* furnizează informaţia despre nodul din calea de mesaje care a generat eroarea. Conţinutul marcatorului *Node* va fi URI-ul acelui nod. În exemplul de mai sus nu există specificat *Node*, deoarece entitatea care a generat eroarea a fost ultimul destinatar şi, evident, emiţătorul cunoaşte URI-ul său.

Elementul *Node* este completat de marcatorul opţional *Role*, care furnizează informaţii despre ceea ce realiza nodul atunci când a eşuat. Conţinutul elementului *Role* este de obicei un URI identificând operaţia (de obicei, un serviciu Web) cauzatoare a erorii. Partea responsabilă cu rezolvarea erorilor poate determina ce „porţiune" a aplicaţiei a funcţionat greşit. Putem observa, deci, că elementele *Node* şi *Role* sunt utile pentru detectarea cu uşurinţă a erorilor care pot apărea.

Să urmărim în exemplul următor elementul *Detail* în forma specifică a unei aplicații.

```
<env:Detail>
   <err:DetaliiEroare
      xmlns:err="http://www.portocale.info/fault">
      <err:codAchizitInvalid>
         <p:cod>P10-01-GR</p:cod>
      </err:codAchizitInvalid>
   </err:DetaliiEroare>
</env:Detail>
```

Prezența marcatorului *Detail* furnizează informații despre erorile survenite atunci când destinatarul final a încercat procesarea corpului mesajului. Este evident că – atunci când elementul *Detail* există – elementele *Node* și *Role* nu vor mai fi specificate. Același lucru este valabil și invers.

Conținutul elementului *Detail* este dependent de specificul fiecărei aplicații în parte.

5.5. *Maniera de codificare a mesajelor* – SOAP Encoding

Deși scopul acestei cărți nu este de a intra în detalii legate de modelul de date utilizat de protocolul SOAP sau de alte scheme de serializare, vom prezenta în secțiunile următoare câteva concepte care să ajute cititorul să aprofundeze specificațiile oficiale legate de codificări.

Termeni folosiți în procesul de codificare

Pentru o mai bună înțelegere a procesului de codificare în maniera SOAP a datelor transportate, este importantă clarificarea următorilor termeni.

Termenul de valoare (*value*) este folosit pentru a descrie o anumită dată codificată cu ajutorul unui element XML. O dată reprezintă un șir de caractere. Valorile dintr-un mesaj SOAP aparțin întotdeauna unui anumit tip de date (similar variabilelor dintr-un limbaj de programare ca Java ori C).

SOAP face distincție între valorile simple (*simple value*) și valorile compuse (*compound value*).

Tipurile *simple value* nu au elemente componente. Pentru a înțelege ce sunt și cum arată aceste valori simple, să urmărim un paralelism între tipurile *int*, *float* și *string*, așa cum sunt reprezentate într-un limbaj de programare (*e.g.*, Java), și modul de reprezentare al lor în SOAP:

```
int cantit = 33;
<cantit xsi:type="xsd:int">33</cantit>
float pi = 3.14159265;
<pi xsi:type="xsd:float">3.14159265</pi>
java.lang.String sortim = "verde";
<sortim xsi:type="xsd:string">verde</sortim>
```

Se poate remarca faptul că tipurile de date SOAP sunt de fapt cele specificate de recomandarea *XML Schema* a Consorțiului Web.

Pentru cazul *compound value*, o valoare e constituită din mai multe elemente componente între care există o relație. Elementele componente care formează o astfel de valoare compusă pot fi accesate prin diverse metode precum: indicarea poziției în secvența de valori (să ne imaginăm cum accesăm un element pe baza indexului, atunci când folosim un tablou), utilizând o cheie (ca în cazul tablourilor asociative – *hash tables*) sau recurgând la numele elementului respectiv (similar structurilor din C). Un astfel de mecanism este cunoscut sub numele de accesor (*accessor*).

Un exemplu în care se folosesc valori compuse (vom face din nou o paralelă cu anumite tipuri de date din Java) este următorul:

```
int[3] Stoc = {10, 74, 33}; // tablou de numere

<Stoc xsi:type="SOAP-ENC:Array"
      SOAP-ENC:arrayType="xsd:int[3]">
   <val>10</val>
   <val>74</val>
   <val>33</val>
</Stoc>

class Produs // o clasă cu doi membri-dată
{
  public int cantit = 33;
  public java.lang.String sortim = "Portocale";
}
Produs produs = new Produs();

<Produs>
   <cantit xsi:type="xsd:int">33</cantit>
   <sortim xsi:type="xsd:string">Portocale</sortim>
</Produs>
```

În acest exemplu, sunt folosiți accesori ordinali pentru obținerea valorilor din variabila de tip tablou *Stoc*.

Cât privește obiectul Java *produs*, pentru a obține elementele lui componente vom folosi accesori cu nume: *cantit* și *sortim*. Vom reveni ulterior la discuția asupra tipurilor compuse (inclusiv tablouri și structuri).

În SOAP, ca în majoritatea limbajelor de programare, valorile au asociat un anumit tip de dată. Așa cum se observă și din exemplu, mai întâi declarăm instanțe ale tipurilor și mai apoi le atașăm o anumită valoare.

În esență, un accesor reprezintă un element care conține sau permite accesul la o valoare. În exemplul următor, *prenume* este un accesor, iar *Ory* este o valoare:

```
<prenume>Ory</prenume>
```

În codul XML de mai jos avem specificat un tip compus reprezentând o combinație de doi accesori (*nume* și *prenume*) care sunt copiii unui singur accesor (*identitate*).

```
<identitate>
    <nume>Blue</nume>
    <prenume>Ory</prenume>
</identitate>
```

Prin intermediul atributelor *id* și *href,* SOAP definește accesori de tip *single--referenced* sau *multi-referenced*. Un accesor simplu-referențiat (*single-referenced*) nu posedă o identitate, exceptând cazul în care este element-copil al altui element. În exemplul următor, <*adresa*> reprezintă un accesor de tip *single-referenced*:

```
<persoana>
   <identitate nume="Blue Ory">
     <adresa>
    <strada>Portocalelor</strada>
    <localit>Iași</localit>
    <tara>România</tara>
     </adresa>
  </identitate>
</persoana>
```

Conform specificațiilor SOAP, elementelor li se permite să referențieze valorile conținute în alte elemente. În acest caz, elementul are asociat un atribut (*id*) care identifică elementul în cadrul căruia se află data referențiată.

Această posibilitate de a referenția valorile altor elemente este importantă din mai multe motive. În primul rând, există șansa ca dimensiunea mesajului să se micșoreze. Să ne imaginăm că avem de trimis un mesaj care conține toți membrii familiei *Blue*. Documentul XML în cauză ar putea avea o structură precum următoarea:

```
<membru>
  <prenume>Ory</prenume>
  <nume>Blue</nume>
</membru>
<membru>
  <prenume>Grappy</prenume>
  <nume>Blue</nume>
</membru>
<membru>
  <prenume>Kiwy</prenume>
  <nume>Blue</nume>
</membru>
...
```

După cum se poate observa, are loc o duplicare nedorită a datelor. Referințele ne permit minimizarea repetițiilor, deoarece mulți accesori pot referi un același element. Când are loc acest lucru, valoarea este de tip multi-referință (*multi-referenced*).

Folosind această facilitate, noul document XML va avea forma:

```
<nume-fam id="nume-fam">Blue</nume-fam>
<membru>
  <prenume>Ory</prenume>
  <nume href="#nume-fam" />
</membru>
<membru>
  <prenume>Grappy</prenume>
  <nume href="#nume-fam" />
</membru>
<membru>
  <prenume>Kiwy</prenume>
  <nume href="#nume-fam" />
</membru>
...
```

Chiar dacă în exemplul de față nu s-a salvat foarte mult spațiu, să ne gândim la utilitatea acestui mecanism atunci când lucrăm, de exemplu, cu o multitudine de elemente compuse.

Variabilele multi-referință sunt utile și atunci când se serializează un graf sau o colecție de obiecte, multe dintre acestea având referință către același obiect. În acest caz, este important să menținem legăturile astfel încât să putem reconstrui obiectul la deserializare.

Această abordare poate fi folosită, de asemenea, pentru a permite unui accesor să referențieze informații externe resurselor, care nu fac parte din

elementul *Envelope* (date binare, de exemplu). În exemplul următor sunt referite informații conținute în interiorul unui document XML extern:

```
<identitate nume='Blue Grappy'>
  <adresa
    href='http://www.portocale.info/inf.xml#BlueGrappy'
  />
</identitate>
```

Tipuri de date

După cum probabil deja cititorii avizați au remarcat, tipurile de date suportate de codificarea SOAP sunt definite în specificațiile *XML Schema: Datatypes* (a se revedea capitolul 2).

Pot fi folosite tipurile elementare ca *int*, *float*, *string*, enumerări, șiruri de octeți etc., plus tipurile derivate din acestea.

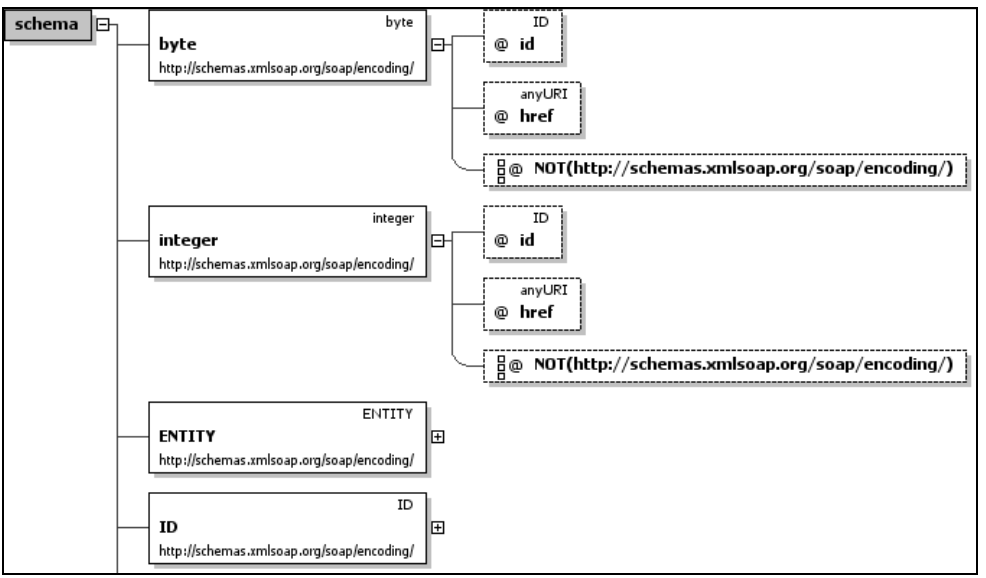


Figura 6. *Fragment din reprezentarea grafică a schemei privitoare la codificare (*SOAP Encoding*)*

Vom face mai jos o trecere rapidă în revistă a acestor tipuri, lăsând la latitudinea cititorului aprofundarea amănuntelor.

• String

Tipul de date *string* este definit de recomandarea *XML Schema*. De remarcat că acest tip *string* nu este identic cu tipul *string* întâlnit în alte limbaje de programare.

În mod particular, interzice apariția unei serii de caractere pe care acele limbaje le permit (sunt caracterele rezervate din XML, precum „<”, „>” ori „&”). Un *string* poate fi codificat ca *single-referenced* sau *multiple-referenced*.

În exemplul de mai jos, „Portocale cu blană” reprezintă o valoare elementară.

```
<string>Portocale cu blană</string>
```

• Enumerare

Specificația *XML Schema: Datatypes* definește un mecanism numit enumerare (*enumeration*). Modelul SOAP îl adoptă în mod direct. Limbajele de programare definesc tipul enumerare într-un mod diferit. În cazul SOAP, *enumeration* – luat drept concept – indică un set distinct de nume. O enumerare particulară reprezintă o listă de valori distincte, apropiate, bazate pe același tip. De exemplu, („1”, „3”, „5”) este o enumerare posibilă bazată pe tipul *integer*. Sunt suportate enumerările pentru toate tipurile simple, exceptând tipul boolean.

Fie următoarea schemă – reprezentarea grafică a acesteia este dată în figura 7:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="lotPortocale" type="xs:int" />
  <xs:element name="greutate" type="xs:float" />
  <xs:element name="culoare">
     <xs:simpleType>
        <xs:restriction base="xs:string">
           <xs:enumeration value="Oranj" />
           <xs:enumeration value="Galben" />
           <xs:enumeration value="Albastru" />
        </xs:restriction>
     </xs:simpleType>
  </xs:element>
</xs:schema>
```

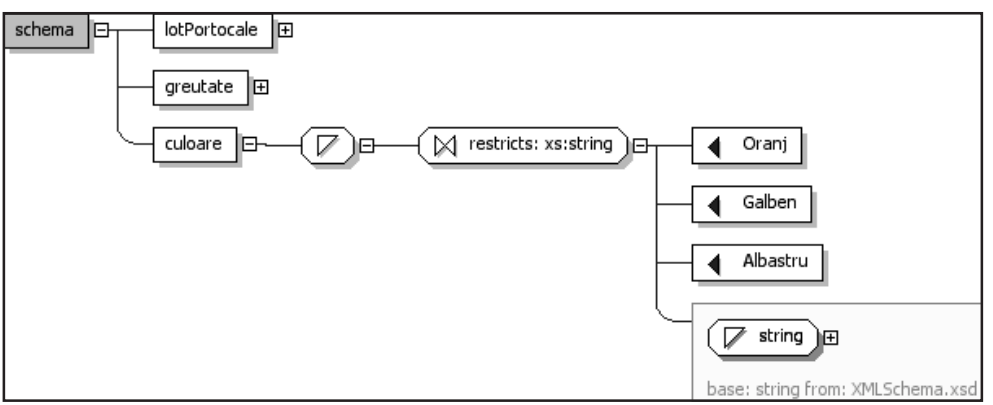


Figura 7. *Reprezentarea grafică a schemei XML specificând o enumerare de elemente*

O posibilă instanță care se conformează schemei XML anterioare poate avea forma:

```
<lotPortocale>12</lotPortocale>
<greutate>3374.01</greutate>
<culoare>Albastru</culoare>
```

• Șir de octeți

Deși în multe limbaje de programare șirurile de caractere (*string*) sunt privite ca fiind șiruri de octeți, în SOAP lucrurile stau diferit. Un *string* este reprezentat cu ajutorului tipului de dată *string*. Astfel, dacă aveți la dispoziție o colecție de octeți care nu reprezintă un *string* (de exemplu, conținutul binar al unei imagini ori al unui fișier executabil), atunci acești octeți trebuie transformați cu algoritmul de codificare *base64* descris în specificația XML Schema.

Un exemplu de codificare *base64* a unui șir de octeți este următorul:

```
<imagine xsi:type="SOAP-ENC:base64">
hsgajTYUTUgsjaguuwq1jgds33ds
</imagine>
```

Un șir de octeți poate fi codificat ca o valoare de tip *single-referenced* sau *multi-referenced*. Regulile de codificare pentru șiruri de octeți sunt similare cu cele pentru *string*.

În particular, un element șir de octeți poate avea un atribut de tip *id*. Elementele accesor suplimentare pot avea asociat atributul *href*.

• Folosirea tipurilor compuse

SOAP definește tipuri corespunzătoare a două modele întâlnite adesea în limbajele de programare.

Primul este *structura*. Un *struct* este o valoare compusă în care numele accesorului permite diferențierea valorilor membrilor și nici un accesor nu are același nume cu un altul (se asigură unicitatea numelor).

Al doilea procedeu recurge la un *tablou*; un *array* permite memorarea unei valori compuse în care indexul (numărul) poziției diferențiază valorile membrilor săi.

Un exemplu în care se folosesc structuri este următorul, considerând fragmentul de schemă XML:

```
<xs:element xmlns:xs='http://www.w3.org/2001/XMLSchema'
            name="Carte">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="autor" type="xs:string" />
```

```
            <xs:element name="titlu" type="xs:string" />
            <xs:element name="pag" type="xs:integer" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Reprezentarea grafică este ilustrată de figura 8.

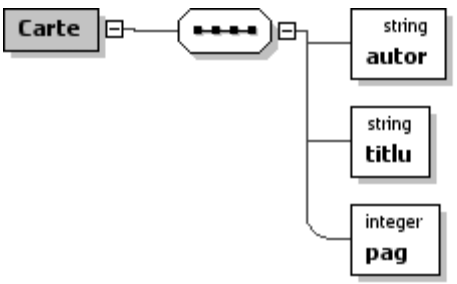


Figura 8. *Reprezentarea grafică a schemei XML specificând o structură*

O structură ce se conformează schemei anterioare poate fi următoarea:

```
<Carte xmlns="urn:infoiasi.ro">
  <autor>Sabin Buraga</autor>
  <titlu>Proiectarea siturilor Web</titlu>
  <pag>346</pag>
</Carte>
```

În cazul tablourilor exemplificăm cu ajutorul schemei de mai jos, care folosește în mod explicit maniera de codificare SOAP:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2001/06/soap-encoding">
    <xs:import
      namespace="http://www.w3.org/2001/06/soap-encoding"
    />
    <xs:element name="numereleFavorite"
                type="enc:Array" />
</xs:schema>
```

Un exemplu de tablou conform schemei de mai sus este:

```
<numereleFavorite
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2001/06/soap-encoding"
    enc:itemType="xs:int" enc:arraySize="2">
    <number>29</number>
    <number>10</number>
</numereleFavorite>
```

5.6. *Reguli de serializare*

Elementele XML într-un mesaj SOAP sunt fie independente, fie imbricate. Din însăși natura XML, majoritatea elementelor sunt imbricate ca sub-elemente ale altora. Marcajele independente nu sunt sub-elemente ale nici unui element, iar în urma serializării ele se găsesc pe primul nivel al arborelui DOM asociat.

Toate valorile într-un mesaj SOAP sunt codificate și incluse în cadrul unui element. Aceasta nu înseamnă că orice element XML conține o valoare. De exemplu, tipurile de date compuse precum structurile sau tablourile includ sub-elemente conținând valori.

Modelul de date SOAP

Atributul *encodingStyle* apare atât în blocurile antet, cât și în cadrul elementului *Body* al unui mesaj SOAP. Acest atribut impune ca mesajul să aibă o anumită formă care se supune regulilor unei scheme XML ce este cunoscută atât de emițător, cât și de receptor.

Din anumite motive, poate exista și riscul ca emițătorul și receptorul să nu aibă acces la aceeași schemă. Pentru evitarea problemelor de acest gen, specificațiile SOAP prevăd existența unei scheme și a unor reguli de codificare proprii. Acestea sunt cunoscute sub denumirea de *SOAP Encoding* și spațiul de nume corespunzător se precizează prin construcția:

```
env:encodingStyle=
  "http://www.w3.org/2003/05/soap-encoding"
```

Regulile pentru codificarea datelor aplicației în mesaje SOAP se regăsesc în recomandarea SOAP sub denumirea de model de date (*SOAP Data Model*). În această parte a specificației se arată cum sunt reduse structurile de date la un graf etichetat.

Mecanismul general este prezentat în figura 9.

Există două aspecte ale codificării. Primul constă în transformarea structurilor de date într-o formă convenabilă de exprimare în XML, cu ajutorul regulilor specificate în modelul de date SOAP. Un alt aspect presupune verificarea datelor existente în documentul XML, astfel încât să respecte regulile din schema SOAP.

În mod obișnuit, instrumentele SOAP vor suporta serializarea și deserializarea grafurilor de obiecte folosind acest model, cu minim de efort din partea programatorului.

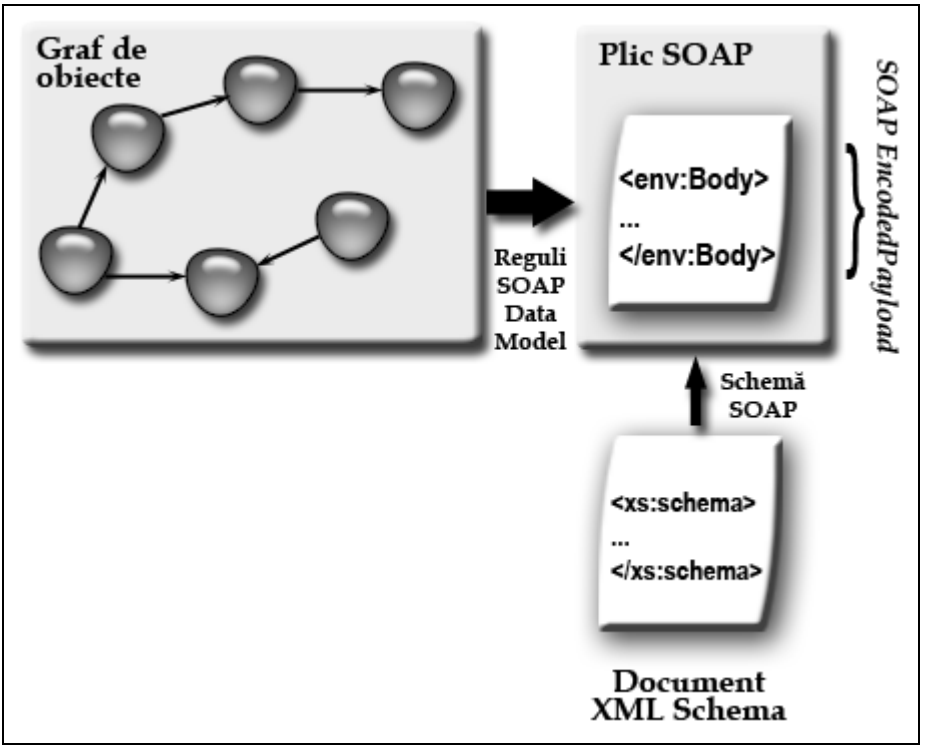


Figura 9. *Mecanismul de codificare oferit de SOAP*

Utilizarea altor codificări SOAP

Multe dintre aplicațiile existente au fost create astfel încât să poată folosi XML. Dacă se dorește utilizarea protocolului SOAP în cadrul acestor aplicații, ar trebui să se recurgă la schemele de modelare deja existente.

Aceasta se poate realiza prin intermediul atributului *encodingStyle*. Deci, în loc să se codifice corpul sau antetul unui mesaj folosindu-se *SOAP Data Model*, se vor putea utiliza regulile din schemele de validare specifice fiecărei aplicații. Evident, trebuie să ne asigurăm că entitatea care dorește serviciile pe care le oferim înțelege modul nostru de codificare.

Un asemenea stil de codificare adaugă un plus de forță protocolului SOAP. Cu excepția faptului că acest mod de codificare favorizează reutilizarea schemelor deja existente și, mai mult, se are de-a face cu schimburi de mesaje și nu cu apeluri de proceduri la distanță (RPC), se încurajează proiectarea de servicii Web care permit o granularitate mult mai mare a interacțiunilor. În esență, se face trecerea de la modelul RPC – care încurajează un model de tip *fine-grain* (un client trimite o serie de date, apoi primește alte date și comunicarea continuă

până când clientul este satisfăcut) – la un model *coarser-grained* (clientul transmite toate datele necesare, astfel încât receptorul, după ce realizează operațiile preconizate, va trimite după un anumit timp răspunsul complet).

Pentru a înțelege mai bine această trecere de la *fine-grained* la *coarser-grained*, să ne imaginăm diferența dintre o discuție la telefon și trimiterea unui fax. În primul caz, purtăm un dialog, în care partenerii joacă un anumit rol, schimbând pe o anumită perioadă de timp tot felul de informații. În al doilea caz însă, cineva care dorește să rezolve o problemă trimite un fax prin intermediul căruia transmite toate informațiile de care cealaltă parte are nevoie.

Am făcut mai sus precizarea că, folosind propriile noastre scheme de codificare, trebuie să scriem și propriul cod cu ajutorul căruia să se poată realiza managementul mesajelor codificate în acest mod. Dacă un anumit nod folosește SOAP RPC (unde codificarea este dată de modelul de date SOAP) și dorim să comunicăm cu acel nod, atunci trebuie să efectuăm o adaptare a instrumentelor noastre în acest sens.

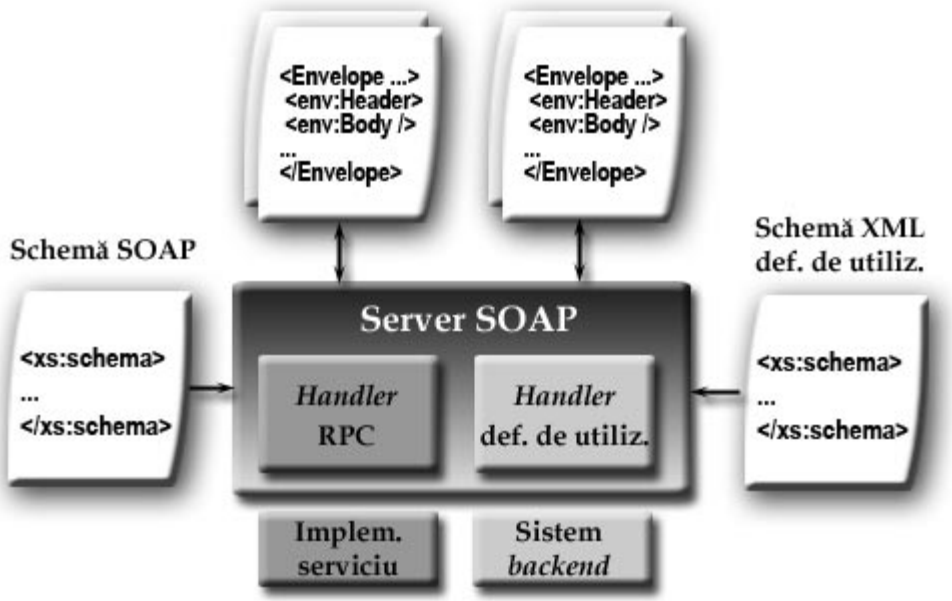


Figura 10. *Procesarea datelor folosind mai multe scheme*

După cum se remarcă în figura 10, *handler*-ul aflat pe serverul SOAP este cel care oferă posibilitatea de înțelegere a mesajelor care sunt codificate cu o altă schemă, diferită de cea implicită, pusă la dispoziție de protocolul SOAP.

6. Maniera de transport al mesajelor SOAP

Așa cum precizam la începutul capitolului de față, în cadrul stivei tehnologiilor care stau la baza serviciilor Web, SOAP se regăsește la un nivel superior nivelurilor de rețea și transport ale Internetului. Pentru aplicațiile folosind SOAP, nu contează ce protocol de transport este utilizat la realizarea schimbului de mesaje. Acest aspect conferă protocolului SOAP o flexibilitate sporită.

În cadrul specificațiile SOAP întâlnim expresia *SOAP Binding*, care precizează modul prin care mesajele SOAP pot fi transmise de la un nod la altul, folosind la bază un anumit protocol (HTTP, SMTP etc.).

Un mesaj SOAP e definit respectând regulile *XML Infoset*. Pentru transportul unui astfel de document, se recurge la un protocol care să permită forma serializată a documentului și care să faciliteze transportul între nodurile din ruta (calea) de mesaje fără pierdere de informații – se asigură fiabilitatea transmisiei de date.

În majoritatea situațiilor se pot utiliza facilitățile native ale protocolului de bază sau, după caz, se pot introduce anumite blocuri antet care să ofere funcționalitățile de care este nevoie.

6.1. *Relația dintre SOAP și HTTP*

Legătura SOAP-HTTP oferă avantajul de a folosi caracteristicile protocolului SOAP împreună cu bogatele trăsături oferite de HTTP. Transportând mesajele SOAP via HTTP nu înseamnă că protocolul SOAP suprascrie semantica existentă a HTTP-ului, ci, mai degrabă, faptul că semantica SOAP se integrează în mod natural în cea a protocolului HTTP.

SOAP, în mod firesc, urmează modelul HTTP cerere/răspuns, cererea SOAP încapsulându-și parametrii în cererea HTTP, iar furnizarea răspunsului SOAP se realizează printr-un răspuns HTTP.

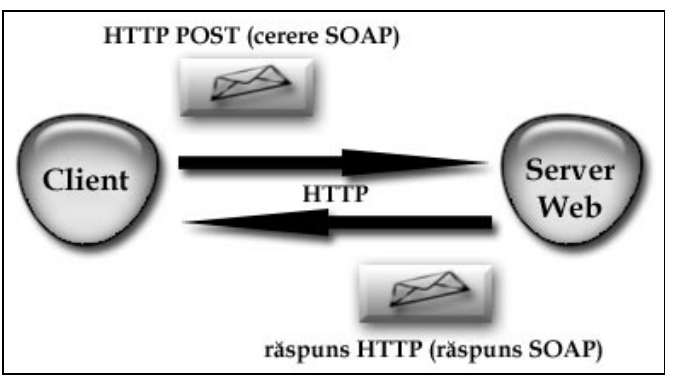


Figura 11. *Încapsularea mesajelor SOAP în cadrul mesajelor HTTP*

De observat că intermediarii SOAP nu sunt aceeași cu intermediarii HTTP. Astfel, un intermediar HTTP adresat cu un câmp din antetul unui mesaj HTTP nu poate controla și prelucra elementul de conținut al mesajului SOAP transportat în cererea HTTP.

Există două modalități de schimb al mesajelor SOAP folosind HTTP, denumite și șabloane de interschimb al mesajelor – MEP (*Message Exchange Pattern*):

- utilizând metoda POST pentru transportul mesajelor SOAP în corpul cererii HTTP, respectiv în mesajul de răspuns (acesta este modelul de schimb al mesajelor SOAP de tip cerere-răspuns – *SOAP Request-Response Message Exchange Pattern*);
- utilizând metoda GET pentru a întoarce mesajul SOAP în corpul mesajului de răspuns HTTP (acesta este modelul de schimb al mesajelor SOAP de tip răspuns – *SOAP Response Message Exchange Pattern*).

Ar putea apărea întrebarea: care metodă ar trebui folosită la un moment dat? Conform modelului arhitectural al spațiului Web, metoda GET poate fi folosită atunci când scopul schimbului de mesaje este de obținere a unei informații, iar resursa cu care s-a interacționat a rămas neschimbată. Astfel de interacțiuni sunt cunoscute în specificațiile HTTP ca fiind sigure (*safe*) și idempotente. Metoda POST poate fi folosită în toate cazurile.

Relația SOAP – HTTP GET

Utilizarea metodei GET semnifică faptul că răspunsul la cererea GET a unui nod SOAP este un mesaj SOAP încorporat în răspunsul HTTP.

Să urmărim exemplul următor în care avem o cerere GET (după cum se observă, parametrii de intrare transmiși serviciului Web sunt incluși în URL, ca șir de interogare – *query string* – prefixat de „?”):

```
GET /Informatii?tip=3307 HTTP/1.1
Host: www.portocale.info
Accept: application/soap+xml
```

Facem observația că în acest caz corpul mesajului e vid (*empty body*).

Câmpul de antet *Accept* este folosit pentru a indica formatul preferat corespunzător cererii realizate. În cazul nostru, este tipul MIME *application/soap+xml*, datele fiind procesate de aplicația client – în loc de tipul *text/html* folosit pentru redarea conținutul în cadrul navigatorului (deci datele în loc să fie destinate „consumului” uman vor fi utilizate pentru o anumită procesare de către o aplicație la nivelul mașinii-client).

Răspunsul cererii GET de mai sus va putea fi următorul:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: NNNN

<?xml version='1.0' ?>
<env:Envelope
   xmlns:env="http://www.w3.org/2002/12/soap-envelope">
 <env:Header>
  ...
 </env:Header>
 <env:Body>
   <!-- diferite informaţii specifice aplicaţiei -->
 </env:Body>
</env:Envelope>
```

Relaţia SOAP – HTTP POST

După cum am precizat anterior, POST este disponibilă pentru toate aplicaţiile, indiferent dacă se „scufundă" date XML în mesajele SOAP sau se utilizează stilul SOAP RPC.

Vom considera mai jos un exemplu în care o cerere SOAP, numită *GetOrangePrice*, este trimisă serverului. Cererea are un parametru cu numele *OrangeType,* iar răspunsul va întoarce un parametru *Price*. Spaţiul de nume propriu aplicaţiei (în fapt, al serviciului Web) ce va fi invocate pe server este *http://www.portocale.info/*. După cum se poate deduce, dorim să obţinem preţul unui anumit sortiment de portocale.

Cererea SOAP va avea structura:

```
POST /Inf HTTP/1.1
Host: www.portocale.info
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: NNNN

<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle=
    "http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:p="http://www.portocale.info/">
    <p:GetOrangePrice>
      <p:OrangeType>Portocale grecești
          fără coajă</p:OrangeType>
    </p:GetOrangePrice>
  </soap:Body>
</soap:Envelope>
```

Răspunsul SOAP, încapsulat într-un mesaj HTTP, este de forma:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: NNNN

<?xml version="1.0"?>
<soap:Envelope
   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle=
     "http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:p="http://www.portocale.info/">
    <p:GetOrangePriceResponse>
      <p:Price>33</p:Price>
    </p:GetOrangePriceResponse>
  </soap:Body>
</soap:Envelope>
```

În cazul în care apare o eroare la procesarea cererii, se va returna un mesaj SOAP *Fault* de genul:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: NNNN

<?xml version='1.0' ?>
<env:Envelope
   xmlns:env="http://www.w3.org/2002/12/soap-envelope">
  <env:Body>
    <env:Fault>
        ...
   </env:Fault>
 </env:Body>
</env:Envelope>
```

6.2. *Relația dintre SOAP și transportul de mesaje de poștă electronică (e-mail)*

Mesajele SOAP pot fi trimise ca parte text a corpului unui mesaj de e-mail sau drept fișier atașat (*attachment*) acestui mesaj.

Facem încă de la început observația că exemplele oferite mai jos nu reprezintă o manieră standard, ci doar o modalitate de transport al mesajelor SOAP.

Să urmărim următoarea cerere SOAP transportată cu ajutorul protocolului SMTP (*Simple Mail Transfer Protocol*):

```
From: admin@axiologic.ro
To: admin@portocale.info
Subject: Contract
Date: Fri, 29 Dec 2006 18:33:00 EST
Message-Id: <M56721-9MGHGJS5757@axiologic.ro>
Content-Type: application/soap+xml

<?xml version='1.0' ?>
<env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
     <p:contract
          xmlns:p="http://www.portocale.info/contracts"
          env:role=
     "http://www.w3.org/2003/05/soap-envelope/role/next"
          env:mustUnderstand="true">
         <p:reference>
            uuid:A292310-S238787-S398298
         </reference>
         <p:dateAndTime>
            2006-12-29T18:33:00.000-05:00
         </p:dateAndTime>
     </p:contract>
     ...
</env:Header>
<env:Body>
     ...
</env:Body>
</env:Envelope>
```

Vom presupune că nodul care recepționează un astfel de mesaj are abilitatea de a procesa mesajele SOAP. În plus, și nodul care a emis cererea trebuie să aibă capacitatea de a putea prelucra, de exemplu, un răspuns SOAP, în cazul apariției unei erori.

Răspunsul pentru cererea anterioară ar putea fi de forma:

```
From: admin@portocale.info
To: admin@axiologic.ro
Subject: ContractAx
Date: Fri, 29 Dec 2006 18:45:07 EST
Message-Id: <CPAX6786382A@portocale.info>
In-replyto: <M56721-9MGHGJS5757@axiologic.ro>
Content-Type: application/soap+xml
<?xml version='1.0' ?>
<env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope">
```

```
<env:Header>
   <p:contract
        xmlns:p="http://www.portocale.info/contracts"
        env:role=
   "http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
      <p:reference>
         uuid:A292310-S238787-S398298
      </p:reference>
      <p:dateAndTime>
         2006-12-29T18:45:07.000-05:00
      </p:dateAndTime>
   </p:contract>
   ...
</env:Header>
<env:Body>
   ...
</env:Body>
</env:Envelope>
```

În acest exemplu, *Message-ID*-ul corespunzător mesajului original este transportat în *In-replyto*, care interconectează mesajele de e-mail la nivelul SMTP. Nu se furnizează însă o corelație specifică SOAP între cerere și răspunsul aferent. În exemplificarea de față, aceasta se bazează pe existența blocului antet *contract*. În general, aceste corelații sunt dependente de fiecare aplicație în parte.

Mai multe discuții legate de legătura *SOAP-Email* puteți găsi la adresa *http://www.w3.org/TR/soap12-email*.

7. Apelul de proceduri la distanță via SOAP RPC

Datorită faptului că specificațiile SOAP furnizează suport pentru RPC, mulți programatori au constatat ușurința cu care se pot dezvolta servicii Web bazate pe SOAP RPC. Chiar dacă SOAP RPC nu este paradigma cea mai dominantă pentru SOAP, este ușor de obținut rezultate folosind-o, deoarece majoritatea instrumentelor suportă bine-cunoscutul model RPC.

Avem în figura de mai jos arhitectura generală SOAP-RPC:

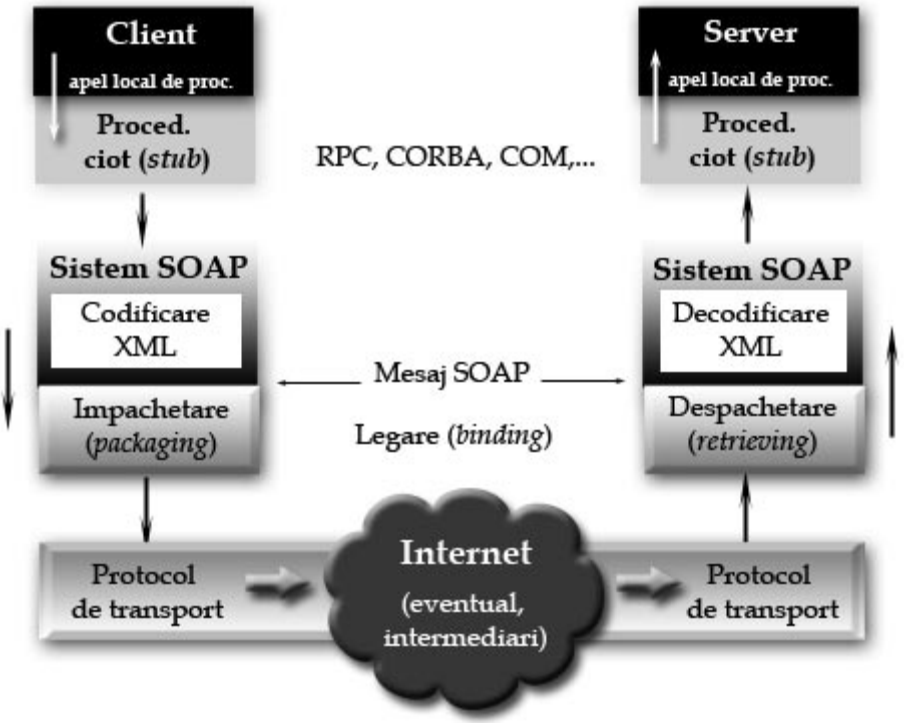


Figura 12. *Mecanismul de transmisie și invocare SOAP RPC*

Conform specificațiilor SOAP, atunci când dorim să realizăm o cerere folosind SOAP RPC trebuie să cunoaștem următoarele:

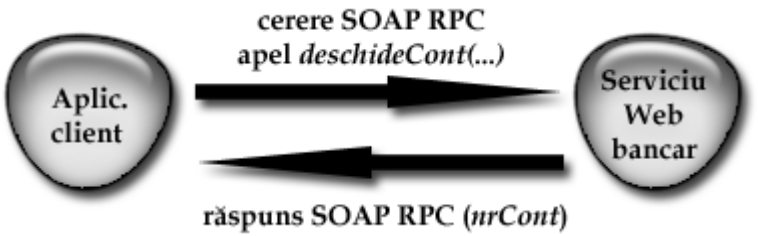
- adresa nodului SOAP țintă;
- numele metodei (procedurii) dorite a fi invocate;
- argumentele (și valorile acestora) care vor fi transmise metodei împreună cu orice parametri de ieșire sau valori de retur;
- modalitatea clară de separare a argumentelor folosite pentru identificarea resursei Web de cele utilizate de către nodul-țintă care face apelul metodei.

Opțional, putem avea date care pot fi transportate în blocurile antet ale mesajului SOAP.

Unul din principalele scopuri ale protocolului SOAP este cel de a încapsula și interschimba apelurile RPC folosind extensibilitatea și flexibilitatea limbajului XML.

SOAP RPC implică două modalități de interschimb al mesajelor: *SOAP Response MEP* și *SOAP Request-Response MEP*, despre care am discutat în cadrul secțiunii 6.1 a prezentului capitol.

Să urmărim în cele de mai jos un exemplu de utilizare a SOAP-RPC. Figura 13 prezintă o interacțiune simplă între un serviciu Web care oferă posibilitatea de deschidere a unui cont bancar și un client (în cazul nostru, compania Axiologic Research) care intenționează să folosească acest serviciu. Serviciul Web pune la dispoziție o operație (metodă) numită *deschideCont(…)*, fiind expus via un server SOAP. Maniera de acces la acest serviciu Web se realizează prin SOAP RPC (protocolul SOAP nu furnizează nici o modalitate de descriere a interfețelor, dar – așa cum s-a văzut în capitolul 3 – acest aspect e suplinit de limbajul WSDL).




```
// Implementarea clientului
Banca b = new Banca();
String cont = b.deschideCont
  (tip, denumire, sitWeb);
```

```
// Implementarea serviciului
class Banca {
 public String deschideCont
    (tip, denumire, sitWeb) {
    // operații pt. crearea
    // unui cont bancar
    return nrCont;
}}
```

Figura 13. *Interacțiunea cu un serviciu Web folosind SOAP RPC*

Clientul interacționează cu serviciul printr-o clasă ciot (numită *stub* sau *proxy*) purtând numele de *Banca*, generată uzual în mod automat de către un instrument (puteți scrie și manual propriul dumneavoastră *stub*). Cu ajutorul acestei clase se facilitează împachetarea și despachetarea datelor aplicației în/din mesaje SOAP RPC.

Liniile de mai jos semnifică o cerere SOAP RPC de la un client către un serviciu Web:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope">
<env:Body>
   <b:deschideCont env:encodingStyle=
```

```
        "http://www.w3.org/2002/06/soap-encoding"
      xmlns:b="http://banca.ex.ro/cont"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi=
        "http://www.w3.org/2001/XMLSchema-instance">
      <b:tip xsi:type="xs:string">Firma</b:tip>
      <b:denumire xsi:type="xs:string">
        Axiologic Research S.R.L.
      </b:denumire>
      <b:sitWeb xsi:type="xs:string">
        http://www.axiologic.ro
      </b:sitWeb>
   </b:deschideCont>
</env:Body>
</env:Envelope>
```

După cum se poate remarca, nu este nimic surprinzător într-un mesaj SOAP RPC. Respectând specificațiile, conținutul este inclus într-un element *Body*. Facem observația că în exemplul nostru nu avem blocuri antet, care nu ne sunt utile în acest caz, dar SOAP RPC nu exclude astfel de elemente.

Revenind la exemplu, vedem că numele elementului *deschideCont* e identic cu numele metodei care va fi apelată de serviciul Web. Conținutul marcatorului *b:deschideCont* va corespunde parametrilor metodei *deschideCont(…)* prezentată în figura de mai sus. Suplimentar, se va mai adăuga atributul *xsi:type* necesar destinatarilor mesajului pentru a converti conținutul fiecărui parametru conform tipurilor de date corespunzătoare limbajului de programare ce implementează serviciul Web în cauză.

Răspunsul la cerere respectă un set mai complex de reguli și convenții:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope">
<env:Body>
   <b:deschideContResponse
        env:encodingStyle=
          "http://www.w3.org/2002/06/soap-encoding"
        xmlns:rpc="http://www.w3.org/2002/06/soap-rpc"
        xmlns:b="http://banca.ex.ro/cont"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi=
          "http://www.w3.org/2001/XMLSchema-instance">
        <rpc:result>b:nrCont</rpc:result>
        <b:nrCont xsi:type="xsd:int">
           56291527
        </b:nrCont>
```

```
    </b:deschideContResponse>
</env:Body>
</env:Envelope>
```

Sesizăm două aspecte legate de acest răspuns care trebuie discutate.

În primul rând, numele elementului de răspuns este același cu cel din mesajul de cerere, numai că i s-a atașat sufixul *Response* (în momentul de față, toate instrumentele folosesc această convenție).

Al doilea aspect constă în faptul că răspunsul este capabil să se muleze pe semantica apelului de procedură pentru multe dintre limbajele de programare actuale. Acest aspect se datorează suportului pentru parametri de tip *in*, *out*, *in/out*. Un parametru de *in* este un parametru de intrare pentru procedura apelată. Un parametru *out* este parametrul care va conține date la terminarea apelului de procedură. Un parametru de tip *in/out* are caracteristicile ambilor parametri *in* și *out*. Diferența constă în faptul că un parametru de tip *out* este similar cu o valoare de retur, dar datele conținute de el pot fi ignorate de către apelantul procedurii.

Datorită importanței pe care o valoare de retur o are în numeroase limbaje de programare, ea este separată de parametrii *in/out*, astfel încât în mesajul nostru se folosește elementul *rpc:result* pentru identificarea elementului ce include răspunsul dorit. În cazul acesta e vorba de *b:nrCont*, elementul care conține valoarea de retur. Alte elemente care nu sunt referențiate sunt tratate ca simpli parametri *in/out*.

Comportamentul este diferit de versiunile anterioare de SOAP, unde valoarea de retur trebuia să fie primul element-copil al răspunsului.

Specificația SOAP 1.2 a fost modificată astfel încât să se rezolve eventuale probleme care puteau apărea (de exemplu, ce se întâmplă în cazul în care nu se întoarce nici o valoare de răspuns).

Este posibil ca și în comunicarea SOAP RPC să apară probleme (excepții). Pentru detectarea și eventual rezolvarea lor, se recurge la avantajul mecanismului furnizat de *Fault*. În plus, au fost adăugate și alte coduri de eroare, al căror spațiu de nume este specificat de adresa *http://www.w3.org/2002/06/soap-rpc* (a se vedea tabelul de mai jos).

Eroare	**Codificarea SOAP privitoare la semnalarea erorii**
O eroare apărută la receptor (de exemplu: *out of memory*)	Trebuie generată o eroare cu valoarea *env:Receiver*
Receptorul nu înțelege codificarea (cazul în care emițătorul folosește altă codificare decât receptorul)	Trebuie generată o eroare în care *Value* are valoarea *env:DataEncodingUnknown*

Eroare	Codificarea SOAP privitoare la semnalarea erorii
Serviciul care a fost invocat nu conţine nici o metodă care să se potrivească cererii RPC	Trebuie generată o eroare în care elementul *Code* conţine elementul *Value* cu valoarea *env:Sender*, iar elementul *SubCode* va avea *Value* setat pe *rpc:ProcedureNotPresent*
Receptorul nu poate procesa argumentele trimise (număr incorect de parametri ori nu se potrivesc tipurile de date)	O eroare în care elementul *Code* conţine *Value* având valoarea *env:Sender* şi elementul *SubCode* include *Value* cu *rpc:BadArguments*

În exemplul de mai jos avem un caz în care a apărut o eroare SOAP RPC. Această situaţie este rezultatul unei cereri construite incorect (nu s-a transmis un parametru). Serviciul Web trimite un mesaj de răspuns în care se poate depista cauza apariţiei erorii. Se observă că ne regăsim în ultimul caz expus în tabelul de mai sus, în care mesajul de răspuns specifică faptul că a apărut o eroare generată de emiţător (`<env:Value>env:Sender </env:Value>`) şi care s-a datorat trimiterii unui număr nepotrivit de parametri *(*`<env:Value>rpc:BadArguments </env:Value>`).

```
<env:Envelope
   xmlns:env="http://www.w3.org/2002/06/soap-envelope"
   xmlns:rpc="http://www.w3.org/2002/06/soap-rpc">
<env:Body>
   <env:Fault>
      <env:Code>
         <env:Value>env:Sender</env:Value>
         <env:Subcode>
            <env:Value>rpc:BadArguments</env:Value>
         </env:Subcode>
      </env:Code>
      <env:Reason xml:lang="ro">
         Lipseşte parametrul &quot;tip&quot;.
      </env:Reason>
   </env:Fault>
</env:Body>
</env:Envelope>
```

Remarcăm, de asemenea, că în mesaj este utilizat şi elementul *Reason* ce include explicaţii în limbaj natural despre eroarea apărută.

8. Versiuni SOAP

O observație interesantă despre SOAP este aceea că elementul *Envelope* nu expune o versiune explicită a protocolului, așa cum fac alte protocoale – un exemplu este HTTP (*e.g.*, HTTP/1.0 *versus* HTTP/1.1), iar un altul este WDDX (`<wddxPacket version="1.0">...</wddxPacket>`).

Proiectanții protocolului SOAP au ales cu bună știință această variantă, deoarece s-a observat că suportul oferit de numărul versiunii este destul de fragil.

Astfel, s-a apelat la utilizarea spațiilor de nume XML: versiunea protocolului este indicată de URI-ul spațiului de nume corespunzător elementului *Envelope*.

Motoarele de căutare/regăsire a serviciilor Web vor cunoaște cum să trateze diferite mesaje SOAP cu care ar putea intra în contact conform următoarelor reguli:

- dacă mesajul are o versiune similară cu aceea utilizată de aplicație, atunci mesajul se va putea procesa;
- dacă mesajul are o versiune mai veche decât orice versiune pe care aplicația știe să o proceseze, atunci pot avea loc două acțiuni: se generează o eroare de tipul *Version Mismatch* și/sau se trimit clientului informații legate de versiunile pe care le poate accepta;
- dacă mesajul are o versiune mai nouă decât toate versiunile pe care programul în cauză știe să le prelucreze, atunci se poate alege procesarea mesajului (ceea ce nu este întotdeauna o soluție bună) sau se poate încerca să se trimită clientului informații privitoare la versiunile pe care le suportă.

Pe ansamblu, mecanismul versiunilor bazate pe URI-uri oferă o mare flexibilitate și adaugă o mai mare putere de conciliere între aplicații în cazul apariției mesajelor cu versiuni diferite.

9. Intermediarii SOAP

SOAP se caracterizează prin extensibilitate. Apare o extensibilitate pe verticală care este conferită de anteturile SOAP. Mai exact, extensibilitatea verticală se referă la capacitatea de a introduce noi informații în cadrul mesajelor SOAP.

Există și o extensibilitate orizontală care se referă la posibilitatea ca diferite părți dintr-un mesaj să poată fi procesate de noduri diferite. Această extensibilitate orizontală este furnizată de către intermediarii SOAP.

Necesitatea existenței intermediarilor

Intermediarii SOAP sunt aplicații care pot procesa părți ale unui mesaj SOAP pe drumul pe care acesta îl parcurge de la emițător la destinația lui finală. Intermediarii pot accepta și pot retrimite mesajele SOAP. Există trei motive principale care fac necesară prezența intermediarilor SOAP: traversarea de domenii de încredere (*crossing trust domains*), asigurarea scalabilității și furnizarea unor servicii suplimentare de-a lungul căii (rutei) de transport al mesajelor.

Modul de traversare a mesajului SOAP prin domenii sigure este de fapt o problemă care se regăsește în cadrul sistemelor distribuite. Să considerăm relația dintre o organizație și Internet. În cadrul organizației, calculatoarele formând o rețea locală constituie un domeniu sigur, însă Internetul este văzut ca un domeniu nesigur. Între aceste domenii există ziduri de protecție (*firewall*-uri) și porți (*gateway*-uri) având rolul de a permite sau nu accesul în „domeniul sigur" al organizației.

Un alt motiv al prezenței intermediarilor se datorează necesității de asigurare a scalabilității sistemelor distribuite. O imagine simplificată a sistemelor distribuite e formată din două tipuri de noduri (gazde): unele care formulează cereri (clienții) și altele care furnizează răspunsul la aceste cereri (serverele). Folosind acest model, se poate crea un sistem distribuit scalabil, cu oricâte servere la un moment dat. Pe acest principiu funcționează și spațiul Web.

Să luăm exemplul unui mesaj de e-mail. Avem două companii *A* și *B*. Corporația *A* trimite un mesaj de e-mail la *B*, însă serverul de e-mail al acesteia este supraîncărcat. În aceste condiții, folosind prioritatea mesajelor și comparând cam cât este de încărcat serverul, mesajul poate să treacă printr-o serie de noduri intermediare înainte de a ajunge la compania *B*. Eventual, mesajul va fi stocat temporar pe una dintre mașinile intermediare.

Este clar că într-un astfel de sistem se cere o dirijare a mesajelor care să nu se bazeze doar pe parametrii mesajului (emițătorul, receptorul, prioritatea), ci trebuie să depindă și de aspecte ca disponibilitatea și încărcarea nodurilor, congestia datelor etc. Nodurile intermediare sunt cele care ascund față de emițător și receptor toate aceste entități prin care „navighează" un mesaj.

Un ultim argument în favoarea existenței nodurilor intermediare se referă la capacitatea acestora de a furniza servicii suplimentare. Unele dintre aceste servicii pot fi foarte importante pentru aplicația finală. Să analizăm câteva exemple:

– schimbul de mesaje într-un mediu nesigur (*e.g.*, Internet). Pentru a fi siguri că mesajul nu va fi alterat, mai întâi putem trimite mesajul unui intermediar

care-l criptează. La recepție, un alt nod intermediar verifică semnătura digitală atașată și dacă totul este în regulă va decripta mesajul;
– asocierea unor facilități legate de drumul pe care un mesaj îl va parcurge. Folosind aceste facilități, mesajul va fi dirijat să treacă exact prin anumiți intermediari, pentru a ajunge la destinație în timpul și forma potrivită. Astfel de informații sunt indispensabile pentru realizarea unor activități precum asigurarea calității serviciilor (QoS – *Quality of Service*), identificarea congestiilor de date etc.

Tipuri de intermediari

SOAP oferă soluții flexibile la trei din cele mai importante probleme legate de nodurile intermediare, răspunzând la următoarele întrebări:

- Cum se pot transmite intermediarilor informațiile necesare?
- Cum se poate identifica entitatea care procesează datele și ce anume procesează?
- Ce se întâmplă cu informația prelucrată de intermediari?

Există două tipuri de intermediari:

- care doar retransmit mesajul primit (*forwarding intermediaries*);
- care modifică și apoi retransmit mesajul (*active intermediaries*).

Intermediarii din prima categorie pot realiza următoarele operații:

- șterg toate blocurile antet procesate;
- elimină toate blocurile antet care nu mai pot fi retrimise și care au fost ignorate atunci când un nod-țintă a procesat mesajul;
- păstrează toate blocurile antet care pot fi retransmise și care nu au fost procesate de nodurile-țintă.

Intermediarii activi îndeplinesc diferite acțiuni care pot modifica traseul mesajului SOAP (uneori în moduri care nu sunt descrise în blocurile antet ale mesajului). Ca exemple de astfel de acțiuni putem menționa apelul unor servicii de securitate, invocarea unor servicii care manipulează conținutul mesajului etc. Acțiunile pe care un astfel de intermediar le efectuează trebuie să poată fi detectate de nodurile care vor primi mesajul.

Facem observația că informația destinată nodurilor intermediare (care se găsește în antet) nu are în general nici o legătură cu informația conținută în corpul mesajului.

10. O privire asupra SOAP 1.1. O comparație cu SOAP 1.2

În acest moment, o multitudine de servicii Web încă folosesc SOAP versiunea 1.1. Vom discuta în continuare ce diferențe și asemănări există între cele două versiuni.

SOAP 1.1 se bazează pe XML 1.0 (documente XML text), iar SOAP 1.2 se fundamentează pe *XML Infoset*, o viziune mai abstractă, arborescentă, a tipurilor de construcții XML. De asemenea, reprezentările interne bazate pe *XML Infoset* pot conduce la optimizări în ceea ce privește performanța.

De asemenea, în SOAP 1.2 au fost schimbate spațiile de nume XML pentru *Envelope* și pentru schema de codificare. SOAP 1.2 nu permite existența a nici unui element după partea de corp a mesajului, pe când definiția schemei pentru SOAP 1.1 permitea o astfel de posibilitate – vezi tabelul de mai jos.

SOAP 1.1	SOAP 1.2
<Envelope> <Header> Zero sau mai multe blocuri antet </Header> <Body> Corpul mesajului </Body> Alte elemente </Envelope>	<Envelope> <Header> Zero sau mai multe blocuri antet </Header> <Body> Corpul mesajului </Body> </Envelope>

Organizația de standardizare a inter-operabilității serviciilor Web – WS-I (*Web Services Interoperability*) – a dezaprobat existența vreunui element suplimentar după *Body*, motivul fiind problemele de incompatibilitate ce ar putea surveni.

În SOAP 1.2 se definește un element antet *Misunderstood*, în interiorul căruia se găsesc informații suplimentare despre erorile care au apărut.

În exemplul următor, receptorul raportează că nu poate înțelege două anteturi obligatorii – *o:oranges* și *p:peaches*.

```
<e:Envelope>
<e:Header>
   <f:Misunderstood qname="o:oranges" />
   <f:Misunderstood qname="p:peaches" />
</e:Header>
<e:Body>
   <e:Fault>
```

```
    <e:Code>
       <e:Value>e:MustUnderstand</e:Value>
    </e:Code>
     </e:Fault>
</e:Body>
</e:Envelope>
```

În SOAP 1.1 se întoarce codul de eroare, dar nici un alt detaliu. De asemenea, în SOAP 1.2 atributul *mustUnderstand* ia valorile logice *true* sau *false*, în timp ce în SOAP 1.1 valorile permise erau 0 sau 1.

Versiunea 1.2 a înlocuit atributul *actor* cu atributul *role*, în esență semantica lor fiind asemănătoare.

În SOAP 1.2, actorului *Next* din SOAP 1.1 i s-au atașat încă două opțiuni:

- *None* – pentru blocurile antet care nu vor fi niciodată procesate în mod direct;
- *Ultimate Receiver* – pentru blocurile antet care vor fi procesate de nodul destinatar final.

Noua manieră de semnalare a erorilor în SOAP 1.2

În ceea ce privește semnalarea erorilor, specificația SOAP 1.2 definește noul cod de eroare *DataEncodingUnknown*, care trebuie generat atunci când un mesaj primit folosește o valoare nerecunoscută, corespunzătoare atributului *encodingStyle*. Codul de eroare *Client* din SOAP 1.1 este redenumit *Sender* în SOAP 1.2, iar codul de eroare *Server* din SOAP 1.1 se regăsește sub numele *Receiver* în noua versiune. Sunt, de asemenea, adăugate noi coduri de eroare pentru RPC (a se revedea subcapitolul 7).

SOAP 1.2 aduce o manieră diferită de structurare a elementelor privitoare la erori – vezi tabelul alăturat.

SOAP 1.1	SOAP 1.2
`Fault`	`Fault`

`faultcode`	`Code`
	`Subcode`
	`Value`

`faultstring`	`Reason`
`faultactor`	`Node` `Role`
`detail`	`Detail`

Dacă SOAP 1.1 permite o extensie a codurilor de eroare folosind notația cu „.” (*dot notation*), versiunea 1.2 înlocuiește această notație cu o reprezentare XML.

SOAP 1.1	SOAP 1.2
<e:Fault> <faultcode> e:Server.Memory </faultcode> <faultstring> Out of memory. </faultstring> </e:Fault>	<e:Fault> <e:Code> <e:Value> e:Receiver </e:Value> <e:Subcode> <e:Value> Memory </e:Value> </e:Subcode> </e:Code> <e:Reason> Out of memory. </e:Reason> </e:Fault>

Modelul de procesare

SOAP 1.2 clarifică modelul de procesare din SOAP 1.1. Astfel, un nod SOAP trebuie să proceseze un mesaj respectând următoarele reguli, în această ordine:

- determinarea rolului pe care îl are de „jucat". Modalitatea de procesare depinde de valorile atributului *role*;
- identificarea antetelor obligatorii corespunzătoare nodului-țintă. Astfel, dacă antetul este marcat cu atributul `mustUnderstand="true"` și atributul *role* are o valoare care se potrivește cu nodul-țintă, atunci acesta e obligat să proceseze acel bloc antet;
- generarea erorii *MustUnderstand*, dacă unul sau mai multe anteturi nu sunt înțelese; aceasta se întâmplă în cazul în care nodul nu conține software-ul potrivit pentru procesarea antetului;
- procesarea antetelor, iar – dacă nodul este destinatarul final –prelucrarea, de asemenea, a corpului mesajului.
- dacă există un nod intermediar, acesta va șterge partea din antet care îi era destinată și va retrimite mesajul.

Mecanismul versiunilor

În SOAP 1.2 s-au adăugat noi semantici pentru a putea lucra cu mesaje SOAP de versiuni diferite.

Un nod SOAP 1.2 care primește un mesaj SOAP 1.1 va răspunde cu un mesaj SOAP 1.1 care conține eroarea SOAP 1.1 *Version Mismatch*.

Un nod SOAP 1.2 care primește un mesaj SOAP de o versiune oarecare (inclusiv mesaje din versiunile superioare) va răspunde cu un mesaj (tot de tip SOAP 1.2) conținând eroarea SOAP 1.2 *Version Mismatch* și un antet *Upgrade* cu lista versiunilor pachetelor pe care le suportă.

Un exemplu este următorul:

```
<u:Upgrade
     xmlns:u="http://www.w3.org/2001/12/soap-envelope">
   <envelope qname="ns1:Envelope" xmlns:ns1=
      "http://schemas.xmlsoap.org/soap/envelope/" />
   <envelope qname="ns2:Envelope" xmlns:ns2=
      "http://www.w3.org/2001/12/soap-envelope" />
</u:Upgrade>
```

Maniera de codificare a datelor

Versiunea 1.2 furnizează o manieră similară de codificare cu SOAP 1.1, însă mult simplificată. Vom prezenta în cele ce urmează diferențele cele mai importante.

În SOAP 1.2 atributul *encodingStyle* poate fi folosit ca element-copil doar al elementelor *Body* și *Head* și al elementului *Detail*. În SOAP 1.1, acest atribut este permis pentru orice element din mesaj.

În ceea ce privește valorile de tip *multi-referenced*, pentru SOAP 1.2 acestea pot fi codificate diferit față de versiunea 1.1.

Vom furniza un exemplu:

SOAP 1.1	**SOAP 1.2**

`<e:Carti>`
`  <e:Carte>`
`  <titlu>Proiectarea`
`  siturilor Web</titlu>`
`  <autor`
`  href="#SabinBuraga"`
`  />`
`  </e:Carte>`
`  <e:Carte>`
`  <titlu>Tehnologii`
`  XML</titlu>`
`  <autor`
`  href="#SabinBuraga"`
`  />`
`  </e:Carte>`
`</e:Carti>`
`<autor id="SabinBuraga">`
`  <nume>Sabin Buraga</nume>`
`</autor>`
	`<e:Carti>`
	`  <e:Carte>`
	`  <titlu>Proiectarea`
	`  siturilor Web</titlu>`
	`  <autor`
	`  id="SabinBuraga">`
	`  <nume>`
	`  Sabin Buraga`
	`  </nume>`
	`  </autor>`
	`  </e:Carte>`
	`  <e:Carte>`
	`  <titlu>Tehnologii`
	`  XML</titlu>`
	`  <autor`
	`  ref="SabinBuraga"`
	`  />`
	`  </e:Carte>`
	`</e:Carti>`

Elementele din schema de codificare SOAP diferă între SOAP 1.1 și SOAP 1.2, după cum se poate remarca din tabelul de mai jos:

	SOAP 1.1	**SOAP 1.2**
Atribut	`href`	`ref`
Tip	`uri-reference`	`IDREF`
Exemplu	`#SabinBuraga` `http://www.sit.ro/csb.gif`	`SabinBuraga`

În ceea ce privește tablourile, în SOAP 1.2 nu mai întâlnim matricele rare (*sparce array*). Față de SOAP 1.1, atributul *arrayType* a fost înlocuit cu *itemType* și cu atributul *arraySize*. Să urmărim tabelul:

SOAP 1.1	**SOAP 1.2**

`<numere enc:arrayType="xs:int[2]">`
`<numar>29</numar>`
`<numar>10</numar>`
`</numere>`
	`<numere`
	`enc:itemType="xs:int"`
	`enc:arraySize="2">`
	`<numar>29</numar>`
	`<numar>10</numar>`
	`</numere>`

Specificația SOAP 1.1 include atributul *root* care poate fi utilizat pentru a marca rădăcina schemei/grafului de codificare. În SOAP 1.2 acest atribut este eliminat.

În SOAP 1.2 omiterea unui accesor este echivalentă cu NIL. Semantica accesorului NIL este însă dependentă de aplicație.

Mecanismul RPC

SOAP 1.1 oferă un model de implementare care constă în existența unui nod destinatar SOAP dat sub forma unui URI, iar în cererea SOAP se regăsește logica aplicației. O astfel de abordare implică ascunderea sub „umbrela" unui URI a tuturor resurselor care pot fi oferite de un nod, împiedicându-se astfel accesarea individuală a fiecărei resurse dorite.

Să revenim la exemplul nostru cu portocale și să considerăm că firma cu situl *www.portocale.info* oferă servicii precum: aflarea prețului unui anumit sortiment de portocale, obținerea datelor privind ofertele promoționale etc. Dacă aceste servicii ar fi plasate sub același URI, ele ar fi efectiv ascunse. O astfel de abordare reduce Web-ul la un nivel de transport și neagă caracterul robust și scalabil al arhitecturii spațiului WWW (în stilul REST amintit în primul capitol).

SOAP 1.2 recomandă ca resursele să fie identificate de URI-uri separate. Acest aspect permite ca resursele SOAP să poată fi accesate nu numai via HTTP POST, ci și prin metoda GET.

SOAP 1.2 acceptă utilizarea fie a tablourilor, fie a structurilor pentru reprezentarea răspunsurilor sau apelurilor RPC, în contrast cu SOAP 1.1, care permitea doar folosirea de structuri. Serializarea bazată pe tablouri este utilă în condițiile în care numărul de argumente nu este cunoscut dinainte.

În următorul exemplu avem apelul unei funcții de adunare de numere, care are un număr variabil de argumente:

```
<e:Envelope xmlns:e='...'>
  <e:Body>
    <a:adunaNumere xmlns:a='...' e:encodingStyle='...'>
      <n>29</n>
      <n>15</n>
      ...
      <n>33</n>
    </a:adunaNumere>
  </e:Body>
</e:Envelope>
```

De asemenea, în SOAP 1.2 corpul mesajelor SOAP care folosesc codificarea SOAP pentru RPC trebuie să conțină un singur element-copil, acesta reprezentând apelul sau răspunsul RPC sub formă de structură sau tablou.

În următoarele linii, elementul *Body* este valid în raport cu codificarea SOAP, dar nu este valid când este folosit în conjuncție cu convențiile RPC.

```
<e:Envelope xmlns:e='...'>
  <e:Body>
    <a:adunaNumere xmlns:a='...' e:encodingStyle='...'>
      <n ref='unNumar' />
      <n ref='unNumar' />
    </a:adunaNumere>
    <n id='unNumar'>33</n>
  </e:Body>
</e:Envelope>
```

În cele de mai jos, elementul *Body* va fi valid atât în raport cu codificarea SOAP, cât și atunci când apare în legătură cu convențiile RPC.

```
<e:Envelope xmlns:e='...'>
  <e:Body>
    <a:adunaNumere xmlns:a='...' e:encodingStyle='...'>
      <n id='unNumar'>33</n>
      <n ref='unNumar' />
```

```
      </a:adunaNumere>
   </e:Body>
</e:Envelope>
```

Versiunea 1.2 introduce și noi coduri de eroare specifice RPC: *ProcedureNotPresent* și *BadArguments*.

Referitor la valorile RPC întoarse, SOAP 1.2 adaugă noi convenții pentru reprezentarea acestora:

SOAP 1.1	**SOAP 1.2**

`<e:Body>`
`<b:deschideContResponse>`
`  <b:nrCont>`
`  56291527`
`  </b:nrCont>`
`</b:deschideContResponse>`
`</e:Body>`
	`<e:Body>`
	`<b:deschideContResponse>`
	`  <rpc:result>`
	`  b:nrCont`
	`  </rpc:result>`
	`  <b:nrCont>`
	`  56291527`
	`  </b:nrCont>`
	`</b:deschideContResponse>`
	`</e:Body>`

Legătura cu HTTP

Vom prezenta în rândurile următoare diferențele între legătura HTTP cu SOAP 1.1, respectiv cu SOAP 1.2. Tipul *text/xml* folosit în SOAP 1.1 a fost înlocuit în versiunea 1.2 cu *application/soap+xml*.

În SOAP 1.1 s-a pus problema folosirii doar a metodei POST pentru transportul mesajelor SOAP via HTTP. Versiunea 1.2 oferă suport și pentru utilizarea metodei GET. Metoda GET asigură că acțiunile de cerere sau de răspuns realizate de un nod SOAP sunt sigure și idempotente, fără efecte secundare (*side effects*).

SOAP 1.1 menționa că în antetul HTTP apare atributul *SOAPAction*, care poate fi folosit pentru a preciza scopul mesajului pe baza unui URI. Pentru SOAP 1.2, conținutul acestui atribut poate fi specificat cu ajutorul parametrului opțional *action*, ce poate fi atașat tipului MIME *application/soap+xml*. Valoarea parametrului *action* va fi desemnată printr-un URI absolut, nu neapărat rezolvabil, putând fi folosită de diverse entități (*e.g.*, *firewall*-uri) pentru filtrare.

Suplimentar, SOAP 1.2 oferă o descriere mai detaliată a codurilor de stare HTTP, de exemplu *2xx* sau *4xx*, pentru raportarea posibilelor situații ce pot surveni.

Securitatea în SOAP

SOAP 1.2 a adus clarificări versiunii 1.1 din multe puncte de vedere, însă la nivel de securitate oferă destul de puține noutăți.

Din acest motiv programatorii trebuie să urmeze anumiți pași adiționali și să utilizeze diverse tehnici de criptare pentru protejarea mesajelor SOAP – a se parcurge și cele prezentate în capitolul 7.

De asemenea, dezvoltatorii de instrumente SOAP trebuie să prevadă situațiile în care nodurile SOAP pot primi mesaje cu diferite date eronate. Este recomandat ca nodurile SOAP să aibă capacitatea să evalueze care este nivelul de încredere în emițătorul unui mesaj și în conținutul aferent.

Referințe

Apparao, V. *et al.* (eds.), *XML Protocol (XMLP) Requirements*, W3C Working Group, Boston, 2003: http://www.w3.org/TR/xmlp-reqs

Berners-Lee, T., Fielding, R., Masinter, L. (eds.), *Uniform Resource Identifiers (URI): Generic Syntax*, IETF RFC 2396: http://www.ietf.org/rfc/rfc2396.txt

Biron, P., Malhotra, A. (eds.), *XML Schema Part 2: Datatypes* (Second Edition), W3C Recommendation, Boston, 2004: http://www.w3.org/TR/xmlschema-2/

Box, D. *et al.*, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, Boston, 2000: http://www.w3.org/TR/SOAP/

Box, D., *A Brief History of SOAP*, XML.com: http://xml.com/lpt/a/2001/04/04/soap.html

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006

Cabrera, L., Kurt, C., *Web Services Architecture and Its Specifications*, Microsoft Press, 2005

Dumbill, E., Johnston, J., St. Laurent, S., *Programming Web Services with XML-RPC*, O'Reilly, 2001

Fallside, D., Walmsley, P. (eds.), *XML Schema Part 0: Primer* (Second Edition), W3C Recommendation, Boston, 2004: http://www.w3.org/TR/xmlschema-0/

Fielding, R. *et al.*, *Hypertext Transfer Protocol – HTTP/1.1*, IETF RFC 2616: http://www.ietf.org/rfc/rfc2616.txt

Freed, N., Borenstein, N., *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, IETF RFC 2045: http://www.ietf.org/rfc/rfc2045.txt

Gudgin, M. *et al.* (eds.), *SOAP Version 1.2 Part 1: Messaging Framework*, W3C Recommendation, Boston, 2003: http://www.w3.org/TR/soap12-part1/

Gudgin, M. *et al.* (eds.), *SOAP Version 1.2 Part 2: Adjuncts*, W3C Recommendation, Boston, 2003: http://www.w3.org/TR/soap12-part2/

Gudgin, M. *et al.* (eds.), *Web Services Addressing 1.0 – SOAP Binding*, W3C Candidate Recommendation, Boston, 2005: http://www.w3.org/TR/ws-addr-soap

Guruge, A., *Web Services: Theory and Practice*, Digital Press, 2004
Haas, H. *et al.* (eds.), *SOAP Version 1.2 Specification Assertions and Test Collection*, W3C Recommendation, Boston, 2003: http://www.w3.org/TR/soap12-testcollection
Karmarkar, A., Gudgin, M., Lafon, Y. (eds.), *Resource Representation SOAP Header Block*, W3C Recommendation, Boston, 2005: http://www.w3.org/TR/soap12-rep/
Mitra, N. (ed.), *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation, Boston, 2003: http://www.w3.org/TR/soap12-part0/
Nielsen, H., Ruellan, H., *SOAP 1.2 Attachment Feature*, W3C Working Group Note, Boston, 2004: http://www.w3.org/TR/soap12-af/
Vedamuthu, A. (ed.), *Web Services Description Language (WSDL) Version 2.0 SOAP 1.1 Binding*, W3C Working Draft, Boston, 2006: http://www.w3.org/TR/wsdl20-soap11-binding
Weerawarana, S. *et al.*, *Web Services Platform Architecture*, Prentice Hall PTR, 2005
* * *, *SOAP Builders*: http://www.soapbuilders.org/
* * *, *World Wide Web Consortium*, Boston, 2006: http://www.w3.org/
* * *, *XML-RPC*: http://www.xmlrpc.com/

Capitolul 5

Descoperirea serviciilor

„Raționamentul meu nu trebuie interpretat ca afirmație, ci drept întrebare."
(Niels Bohr)

Descoperirea serviciilor este una dintre cele mai importante probleme dezbătute pe larg în cadrul comunității serviciilor Web. Aceasta, de fapt, poate fi formulată cu ajutorul unei întrebări simple: cum poate afla un client despre existența și modul de utilizare a unui serviciu?

Procesul de descoperire a serviciilor Web poate fi unul centralizat sau distribuit. Există mai multe abordări de soluționare a acestei probleme, care constituie subiectul capitolului de față.

1. Descoperirea serviciilor Web prin UDDI

Cele mai multe aplicații destinate comerțului electronic și care recurg la servicii Web urmează căi divergente pentru conectarea cumpărătorilor, furnizorilor de produse, piețelor și ofertanților de servicii. Fără largi investiții în infrastructura tehnologică, companiile de toate mărimile și tipurile pot tranzacționa afaceri bazate pe Internet doar cu parteneri globali deja cunoscuți și care furnizează aplicații și servicii Web compatibile.

Specificațiile UDDI (*Universal Description, Discovery, and Integration*) au ca scop definirea unui set de servicii care să permită descrierea și descoperirea:

– afacerilor, organizațiilor și a altor furnizori de servicii Web;
– serviciilor Web pe care diverse entități le pun la dispoziție;
– interfețelor cu ajutorul cărora pot fi accesate aceste servicii.

UDDI reprezintă nivelul superior în stiva formată din tehnologii ca TCP/IP, HTTP, XML, XML Schema și SOAP, cu care cititorul este deja familiarizat.

1.1. *Scurt istoric*

Atunci când a apărut UDDI, cea mai mare atenție s-a fixat asupra UBR (*UDDI Business Registry*) – o implementare publică a standardului UDDI, care este de fapt un catalog principal folosit pentru publicarea serviciilor de tip *e-commerce*. Deși UBR rămâne un aspect important în ceea ce privește UDDI, el reprezintă doar o componentă a acestuia.

Tabelul următor sintetizează nivelurile specificațiilor UDDI (standardele disponibile de-a lungul anilor):

Versiunea UDDI	Anul lansării	Obiectiv
1.0	2000	Crearea cadrului pentru cataloage (regiştri) destinate serviciilor de tip *e-business*.
2.0	2001	Alinierea specificațiilor la standardele Web existente şi furnizarea unei taxonomii flexibile a serviciilor.
3.0	2004	O principală caracteristică este cea de adăugare a suportului pentru securitate.

Actualmente este în vigoare UDDI 3.0, bazat pe mai vechiul standard UDDI 2.0 propus de OASIS (*Organization of the Advancement of Structured Information Standards*).

UDDI poate fi considerat drept un meta-serviciu folosit pentru localizarea serviciilor Web, permițând în același timp interogări „robuste", în raport cu cantitatea imensă de meta-date privitoare la serviciile Web oferite de diverși implementatori.

1.2. *Concepte de bază ale UDDI*

Modelul de date UDDI

În figura 1 se pot remarca principalele entități stocate în regiştrii UDDI, suplimentar fiind ilustrate și relațiile dintre ele.

Tipurile de date folosite (care constituie nucleul modelului UDDI) sunt definite cu ajutorul unor scheme XML. A fost ales formatul XML, deoarece este independent de platformă și permite exprimarea într-un mod natural a relațiilor ierarhice.

Schemele XML utilizate de UDDI specifică diverse tipuri de date fundamentale necesare utilizatorilor și aplicațiilor în vederea folosirii unui serviciu Web.

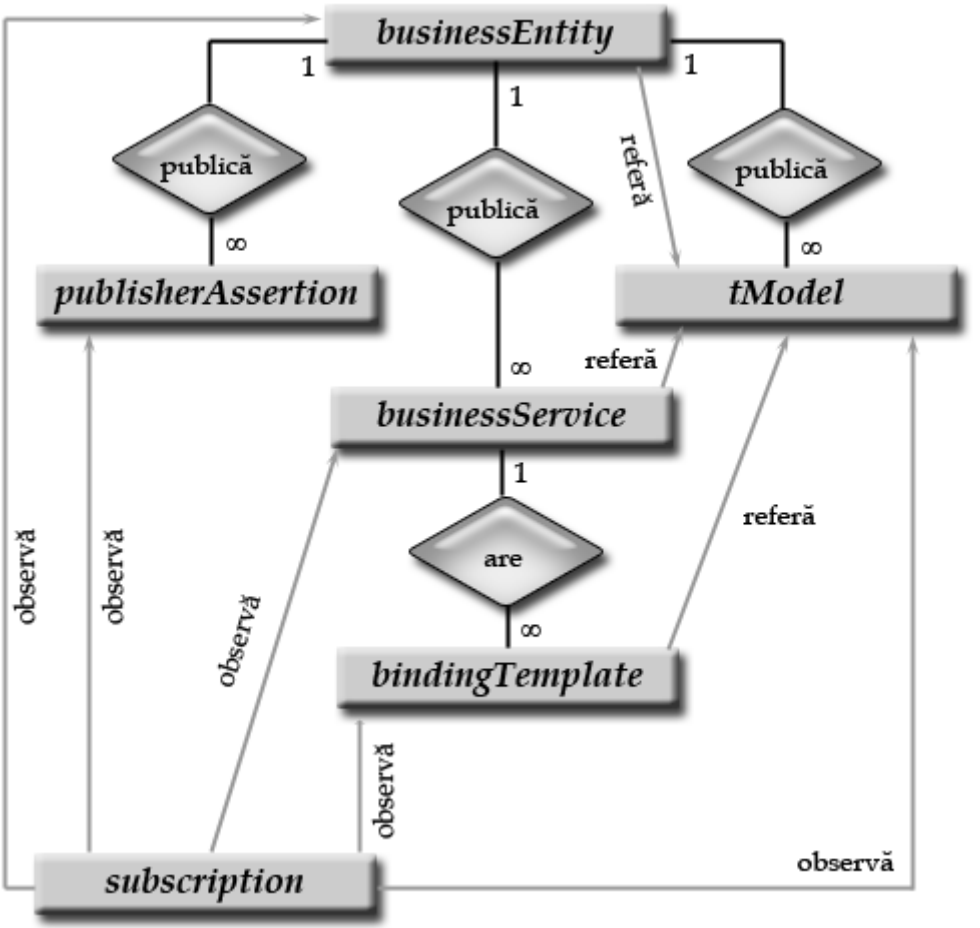


Figura 1. *Tipurile de date UDDI și relațiile stabilite între acestea*

Tipurile de informații de bază sunt următoarele:

- *businessService* – descrie o colecție de servicii Web înrudite, oferite de o organizație desemnată de *businessEntity*;
- *businessEntity* – semnifică informația privitoare la organizația care publică un serviciu;
- *bindingTemplate* – specifică detalii tehnice despre serviciu, incluzând o referință la interfața de programare (API-ul) serviciului;
- *tModel* – descrie un model tehnic (*technical model*) reprezentând un concept reutilizabil, cum ar fi tipul unui serviciu Web, un protocol folosit de serviciile Web, diverse alte atribute sau meta-date (*e.g.*, semnături digitale, relații cu alte servicii etc.).

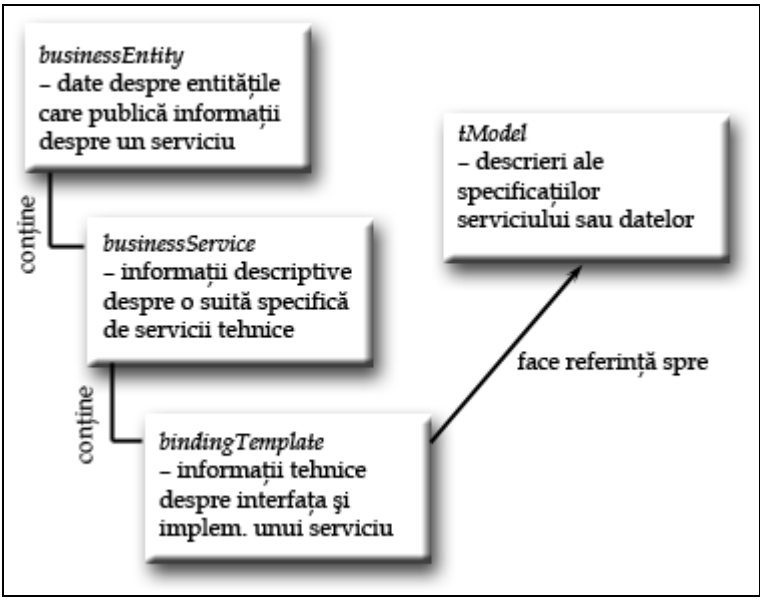

businessEntity
– date despre entitățile care publică informații despre un serviciu
conține
businessService
– informații descriptive despre o suită specifică de servicii tehnice
conține
bindingTemplate
– informații tehnice despre interfața și implem. unui serviciu
tModel
– descrieri ale specificațiilor serviciului sau datelor
face referință spre

Figura 2. *Tipurile de informații de bază UDDI*

O structură de tip *businessEntity* conține una sau mai multe structuri distincte de tip *businessService*. Similar, o structură de tip *businessService* poate include una sau mai multe structuri *bindingTemplate*.

În versiunile recente există încă două tipuri de date care facilitează afilierea regiștrilor:

- *publisherAssertion* – semnifică relațiile dintre entitățile stocate într-un registru;
- *subscription* – menține anumite cereri pentru a se detecta schimbările care pot avea loc în ceea ce privește un set de entități.

Toate aceste tipuri de date sunt stocate persistent de regiștrii UDDI. Pentru un anumit catalog UDDI, structurii de date fundamentale i se asociază un identificator unic, în acord cu o anumită schemă standard. Acest identificator se numește *cheie* (UDDI *key*).

Mai remarcăm și faptul că o anumită structură *businessEntity* (identificată de o cheie unică) va conține întotdeauna o anumită instanță a unei structuri de tip *businessService* (specificată de propria sa cheie).

De oricâte ori este necesar, prin intermediul acestor chei pot fi folosite referințe la o anumită entitate.

Informațiile de tip businessEntity

Fiecare intrare de tip *businessEntity* stochează date descriptive privitoare la o afacere sau organizație și, prin intermediul entităților *businessService* pe care le conține, pune la dispoziție informații despre serviciile oferite.

O entitate *businessService* descrie un serviciu, dintr-un anumit punct de vedere. Similar, fiecare *bindingTemplate* inclus într-o structură *businessService* furnizează o descriere tehnică a serviciului Web care aparține serviciului logic declarat de *businessService*.

Structura *businessEntity* conține următoarele elemente:

– *uddi:discoveryURLs*, care reprezintă o listă de URL-uri care indică spre documente conținând mecanisme de descoperire; avem mai jos un exemplu:

```
<discoveryURLs>
   <discoveryURL useType="businessEntity">
      http://ws.portocale.info?businessKey
        =uddi:portocale.info:registry:Sales:74
   </discoveryURL>
</discoveryURLs>
```

– *uddi:name*, care este un element obligatoriu, semnificând numele procesului global de afacere oferit (de exemplu, managementul ordinelor de plată);
– *uddi:description*, ce oferă o descriere, fiind un element opțional;
– *uddi:contacts*, ce include informații de contact privitoare la entitatea care a publicat serviciul; un exemplu este următorul:

```
<contacts>
   <contact useType="Sales">
      <personName>Blue Ory</personName>
      <email>blue.ory@portocale.info</email>
   </contact>
</contacts>
```

– *uddi:businessServices*, care reprezintă o listă de servicii furnizate de *businessEntity*;
– *uddi:identifierBag* – pe lângă *businessKey*, care identifică în mod unic o structură *businessEntity* într-un registru, acest element opțional permite ca structurile *businessEntity* să poată fi regăsite în acord cu un sistem de identificare public;
– *uddi:categoryBag* – conține o listă de categorii de afaceri – de exemplu, după industrie, după regiunea geografică etc. –, fiecare descriind un anumit aspect conform unei taxonomii alese (*e.g.*, schema de clasificare NAICS – *North American Industry Classification System*);
– *dsig:Signature*, reprezentând o semnătură digitală XML atașată unei entități *businessEntity*.

Notăm faptul că elementele *name*, *description* și *contacts* oferă informații textuale privitoare *businessEntity*, eventual în mai multe limbi.

Orice instanță a elementului *businessEntity* este identificată în mod unic de atributul *businessKey*. Atunci când o structură de tip *businessEntity* este publicată într-un registru UDDI, atributul *businessKey* trebuie omis în cazul în care cel care publică informațiile dorește ca registrul să genereze automat o cheie. Atunci când o entitate de tip *businessEntity* este obținută dintr-un registru UDDI, atributul *businessKey* trebuie să existe obligatoriu.

Informațiile de tip businessService

Elementul *businessService* este o structură reprezentând un serviciu specificat logic și conține informații descriptive în termeni de afaceri. Informații tehnice despre *businessService* se găsesc în structurile de tip *bindingTemplates* (vezi *infra*).

Structura *businessService* încapsulează următoarele elemente:

– *uddi:name*, care semnifică numele serviciului din cadrul procesului de afacere oferit (*e.g.*, recepționarea unui ordin de plată);
– *uddi:description*, care specifică o descriere a serviciului;
– *uddi:bindingTemplates* – este o listă de descrieri tehnice a serviciilor Web furnizate de *businessService*;
– *uddi:categoryBag*, ce conține o lista de clasificări ale afacerilor, fiecare descriind un aspect al elementului *businessService*;
– *dsig:Signature* reprezentând o semnătură digitală asociată entității *businessService*.

Elementului *businessService* îi sunt atașate și atributele:

– *serviceKey*, ce identifică în mod unic o entitate *businessService*;
– *businessKey* reprezentând o cheie referitoare la o intrare *businessEntity*.

Când o informație de tip *businessKey* este inclusă într-un registru UDDI, atributul *serviceKey* poate fi omis dacă organizația care o publică dorește ca registrul să genereze o cheie. Atunci când *businessEntity* este obținut de la un registru, atributul *serviceKey* trebuie să fie prezent.

Atributul *businessKey* identifică în mod unic o informație *businessEntity* care furnizează o structura *businessService*. Fiecare entitate *businessService* este inclusă într-o singură structură *businessEntity*.

Un exemplu de intrare de tip *businessService* este următorul (cheile sunt furnizate sub formă de identificatori unici universali de tip UUID; pentru UDDI versiunea 3, acești identificatori nu trebuie neapărat să fie numerici):

```
<businessEntity ...>
   ...
  <businessServices>
     <businessService
       businessKey="uuid:..." serviceKey="uuid:...">
       <name>Preia un ordin de plată</name>
       <description xml:lang="ro">
          Serviciu pentru recepţionarea
          ordinelor de plată.
        </description>
       <bindingTemplates>
         ...
       </bindingTemplates>
       <categoryBag>
         ...
       </categoryBag>
     </businessService>
  </businessServices>
</businessEntity>
```

Informaţiile de tip bindingTemplate

Entităţile *bindingTemplate* furnizează descrierile tehnice pentru serviciile Web. Un element *bindingTemplate* corespunde unei instanţe a serviciului oferit la o anumită adresă, în general furnizând URI-ul acesteia. De asemenea, acest element face referiri la datele de tip *tModel* (pe care le vom descrie mai jos), precum şi la parametri şi setări specifice aplicaţiei.

Structura *bindingTemplate* conţine următoarele elemente:

– *uddi:description*, care oferă informaţii textuale despre *bindingTemplate*;
– *uddi:accessPoint*, care este un şir de caractere ce permite accesul la serviciul Web descris. În mod obişnuit, este un URI, dar poate fi şi o adresă de e-mail sau chiar un număr de telefon;
– *uddi:hostingRedirector*, ce reprezintă un element folosit doar pentru păstrarea compatibilităţii cu versiunea UDDI 2.0. Funcţionalitatea sa este acoperită de *uddi:accessPoint*;
– *uddi:tModelInstanceDetails*, care semnifică o listă de unul sau mai multe elemente *tModelInstanceInfo*. Colecţia de atribute *tModelKey* existente în elementele *tModelInstanceInfo* formează aşa-numita amprentă tehnică a serviciului Web, care poate fi folosită pentru identificarea serviciilor compatibile.
– *uddi:categoryBag*, ca mai sus, reprezintă o colecţie de categorii asociate serviciului;
– *dsign:Signature*, ce semnifică o semnătură digitală.

Similar sunt specificate și atributele *bindingKey* și *serviceKey*, având aceleași semnificații ca în cazul anterior.

Un exemplu este următorul:

```
<businessService ...>
   <bindingTemplates>
      <bindingTemplate
         serviceKey="uuid:..." bindingKey="uuid:...">
         <accessPoint URLType="http">
            http://ws.portocale.info:3333/Sales
         </accessPoint>
         <tModelInstanceDetails>
            ...
         </tModelInstanceDetails>
      </bindingTemplate>
   </bindingTemplates>
</businessService>
```

1.3. *Structura* tModel

Unul dintre scopurile UDDI este acela de a facilita găsirea cu ușurință a serviciilor Web. Un alt obiectiv este cel de a oferi descrieri complete ale acestor servicii, astfel încât oamenii și programele să poată interacționa ușor cu servicii despre care nu știau mai nimic în prealabil. Aceste scopuri sunt de fapt trăsăturile esențiale ale structurilor *tModel*. Entitățile de bază UDDI sunt tocmai instanțele *tModel*.

La nivel general, scopul elementelor *tModel* este de a furniza un sistem de referință abstract. Există două motive principale pentru utilizarea structurilor *tModel*: determinarea compatibilității între serviciile Web și folosirea lor drept referințe la un spațiu de nume fixat.

Utilizare

Din punct de vedere al utilizării, o instanță a unei structuri *tModel* poate fi referită de mai multe structuri *businessEntity*. De asemenea, în diferite apeluri ale interfeței de programare pot apărea referiri la *tModel*.

Utilizarea cea mai obișnuită a elementelor *tModel* este în reprezentarea specificațiilor tehnice și a conceptelor. De exemplu, un *tModel* poate fi folosit pentru a reprezenta o specificație care definește un protocol de transport *wireless*, formatul interschimburilor de date și secvența de reguli ce trebuie urmate în procesul de schimb al mesajelor.

O aplicație care comunică cu o alta va prefera, desigur, să adere la specificații (convenții) prestabilite. În acest caz, proiectanții acestor specificații pot stabili o identitate tehnică unică într-un registru UDDI, publicând informații despre acestea în cadrul unui *tModel*.

În timp ce principalul motiv pentru înregistrarea unui *tModel* într-un registru UDDI este definirea unei identități, specificația reală sau setul de documente care descriu conceptul *tModel*-ului nu vor face parte din registru, însă vor putea fi referite (cu ajutorul structurii *overviewDoc*). Evident, cei care publică astfel de documente, specificând conceptul pe care un *tModel* îl reprezintă, trebuie să ofere descrieri folosind formate și limbaje bine-cunoscute.

Din moment ce un *tModel* a fost publicat, alte organizații pot considera că un anumit serviciu Web pe care îl furnizează respectă specificațiile acestuia și își afirmă „disponibilitatea" prin simpla incluziune a unei referințe la acel *tModel* (folosindu-se *tModelKey* din structura *bindingTemplate*) – un mod similar este cel în care cineva poate „cita" un document Web, făcând o referire la acesta printr-o legătură hipertext desemnată de un URI.

Această abordare facilitează găsirea serviciilor Web compatibile cu o anumită specificație. Odată ce o valoare validă pentru *tModelKey* este cunoscută, este ușor de descoperit că o anumită entitate particulară de tip *businessEntity* are asociat un serviciu Web referențiind un *tModel*. În acest mod, un *tModelKey* devine așa-numita amprentă tehnică (*technical fingerprint*), unică pentru o anumită specificație.

Elementele *tModel* pot fi incluse în cadrul marcatorilor *identifierBag*, *categoryBag*, *address* și *publisherAssertion* care sunt folosiți pentru specificarea identității organizaționale și a diferitelor categorii. Utilizat în acest context, un *tModel* desemnează un sistem de valori folositor pentru identificarea sau clasificarea entităților UDDI, având astfel un rol în constituirea unor taxonomii referitoare la serviciile Web.

Pentru a reprezenta faptul că o anumită afacere descrisă de elementul *businessEntity* are un anumit identificator ID asociat, *keyedReference* este plasat într-un *identifierBag*. Acest *keyedReference* va avea atributul *keyValue* cu valoarea identificatorului ID dat și referă un *tModel* semnificând „sistemul identificat de ID". Împreună, *keyValue* și referința la *tModel* specifică o valoare particulară într-un anumit sistem de valori.

Structura unui element de tip tModel

Un element *tModel* este compus din următoarele sub-elemente:

– *uddi:name* care asociază un nume unic, obligatoriu (în general este un URI);
– *uddi:description*, reprezentând o descriere textuală opțională;

– *uddi:overviewDoc*, care face referire la informații privitoare la *tModel*; include un element *overviewURL* ce reprezintă adresa la care se găsește documentul descriind *tModel*-ul. Atributul *useType* al marcatorului *overviewURL* poate avea mai multe valori. Dacă valoarea lui este „*text*”, atunci înseamnă că *overviewURL* conține informații textuale. O altă valoare des întâlnită pentru *useType* este *wsdlInterface*, ceea ce semnifică faptul că *overviewURL* referă un document WSDL care poate fi reutilizat de diferite implementări;
– *uddi:identifierBag*, ce conține o listă de identificatori logici, cu rol de meta-date;
– *uddi:categoryBag*, care furnizează o listă de clasificări descriind anumite aspecte ale *tModel*-ului.

Structura *tModel* are asociate și atributele:

- *tModelKey*, care identifică în mod unic un *tModel*;
- *deleted* este un câmp care specifică dacă *tModel*-ul a fost șters logic (valorile sale pot fi *true* sau *false*).

Un exemplu de informații conținute de un *tModel* este următorul, descriind comportamentul unui serviciu Web:

```
<tModel tModelKey="uuid:...">
   <name>Vânzări online de portocale</name>
   <description xml:lang="ro">...</description>
   <overviewDoc>
      http://ws.portocale.info:3333/Sales?WSDL
   </overviewDoc>
   <identifierBag>...</identifierBag>
   <categoryBag>...</categoryBag>
</tModel>
```

1.4. *Informații privitoare la publicarea serviciului*

Structura publisherAssertion

Multe afaceri și organizații nu sunt efectiv reprezentate de o singură entitate de tip *businessEntity*. Atunci când o organizație este desemnată de mai multe structuri *businessEntity*, s-ar dori ca acestea să fie interconectate prin relații vizibile. Aceasta se poate realiza cu ajutorul elementului *publisherAssertion*.

Pentru a elimina posibilitatea ca o entitate care a publicat un serviciu să revendice o relație care nu e reciprocă, ambii parteneri trebuie să facă publice aceleași „afirmații” pentru ca relația lor să devină vizibilă (se asigură astfel reciprocitatea).

Structurii *publisherAssertion* îi corespund următoarele elemente:

- *uddi:fromKey* – identifică instanța „sursă" de tip *BusinessEntity* care stabilește acordul (relația);
- *uddi:toKey* – specifică instanța „destinație" de tip *BusinessEntity* participantă la acord;
- *uddi:keyedReference* – descrie relația dintre cele două părți;
- *dsig:Signature* – desemnează o semnătură digitală XML. Atunci când o structură *publisherAssertion* este semnată, fiecare semnătură digitală trebuie să fie furnizată de propriul element *dsig:Signature*.

Structura operationalInfo

La publicarea unui serviciu, în registrul UDDI sunt stocate și o serie de informații operaționale, incluzând data și timpul la care structura de date a fost creată sau modificată, identificatorul nodului UDDI și identitatea celui care a publicat datele.

Aceste informații operaționale vor fi memorate într-un element o*perationalInfo*, pentru structurile de date de tip *businessEntity*, *businessServices*, *bindingTemplate* și *tModel*.

Sub-elementele care pot fi folosite în cadrul o*perationalInfo* sunt *created*, *modified*, *modifiedIncludingChildren*, *nodeID*, *authorizedName*. Entitatea UDDI care are asociată o structură *operationalInfo* se identifică prin atributul obligatoriu *entityKey*.

1.5. *Taxonomia entităților UDDI*

UDDI furnizează o structură semantică a informațiilor referitoare la serviciile Web dintr-un registru. UDDI permite utilizatorilor să definească taxonomii multiple. Astfel, utilizatorii nu sunt legați de un singur sistem taxonomic, ci pot folosi un număr nelimitat de clasificări dorite.

Specificațiile UDDI includ anumite definiții privitoare la relațiile ierarhice dintre o instanță a unei implementări UDDI și alte entități de care aceasta este legată.

Serverele UDDI sunt împărțite în următoarele tipuri:

- nod (*node*) – este un server UDDI care suportă minimul de funcționalități definite în specificație. Poate executa una sau mai multe operații asupra datelor UDDI la care are acces. Este membrul unui singur registru UDDI;
- registru/catalog (*registry*) – este format din unul sau mai multe noduri. Un registru execută setul complet de funcționalități definit în specificațiile UDDI.

- regiştri afiliaţi (*affiliated registries*) – sunt regiştrii UDDI care implementează reguli de partajare a informaţiilor. Aceşti regiştri afiliaţi partajează un spaţiu de nume comun pentru cheile UDDI care identifică în mod unic înregistrările de date.

În versiunea 3.0, una dintre modificările esenţiale este cea privitoare la conceptul de regiştri afiliaţi. Să urmărim mai jos câteva aspecte referitoare la tipurile de regiştri UDDI:

Tipul registrului	**Descriere**	**Exemplu de aplicaţie**
De tip *enterprise* sau privat (*corporate/ private*)	Un registru intern, al unei întreprinderi, aflat în spatele unui *firewall* izolat de Internet. Accesul la facilităţile administrative şi la datele din regiştri este restricţionat. Datele nu sunt partajate cu alţi regiştri.	Catalog UDDI la nivel de corporaţie (*Enterprise Web Service Registry*)
De tip afiliat (*affiliated*)	Un registru dezvoltat în cadrul unui mediu controlat şi cu acces limitat la clienţi autorizaţi. Funcţiile de administrare pot fi realizate şi de alt partener de încredere. Datele pot fi partajate cu alţi regiştri într-o manieră controlată.	Reţea de parteneri comerciali (*Trading Partener Network*)
Public	Din perspectiva unui utilizator final, un registru public este un serviciu obişnuit. Deşi funcţiile administrative pot fi securizate, accesul la datele din regiştri este public. Datele pot fi partajate sau transferate între diferiţi regiştri, iar conţinutul propriu-zis poate fi sau nu moderat.	Catalog de tip *UDDI Business Registry* (UBR)

Altfel spus, afilierea permite sistemului UDDI să suporte o varietate de topologii de reţele sau de infrastructuri. Structura unui registru UDDI permite reflectarea realităţii existente în interacţiunile reale dintre entităţile *business*.

Managementul versiunilor multiple ale datelor stocate de regiştrii UDDI constituie o reală provocare pentru o astfel de infrastructură distribuită. Specificaţiile UDDI oferă un cadru care să faciliteze menţinerea şi asocierea de chei UDDI înregistrărilor din regiştri, dar nu pun în discuţie existenţa vreunui suport de definire de scenarii *business*. Regulile de afaceri şi scenariile aferente se lasă în responsabilitatea dezvoltatorului software care însă, poate folosi infrastructura de bază oferită de UDDI.

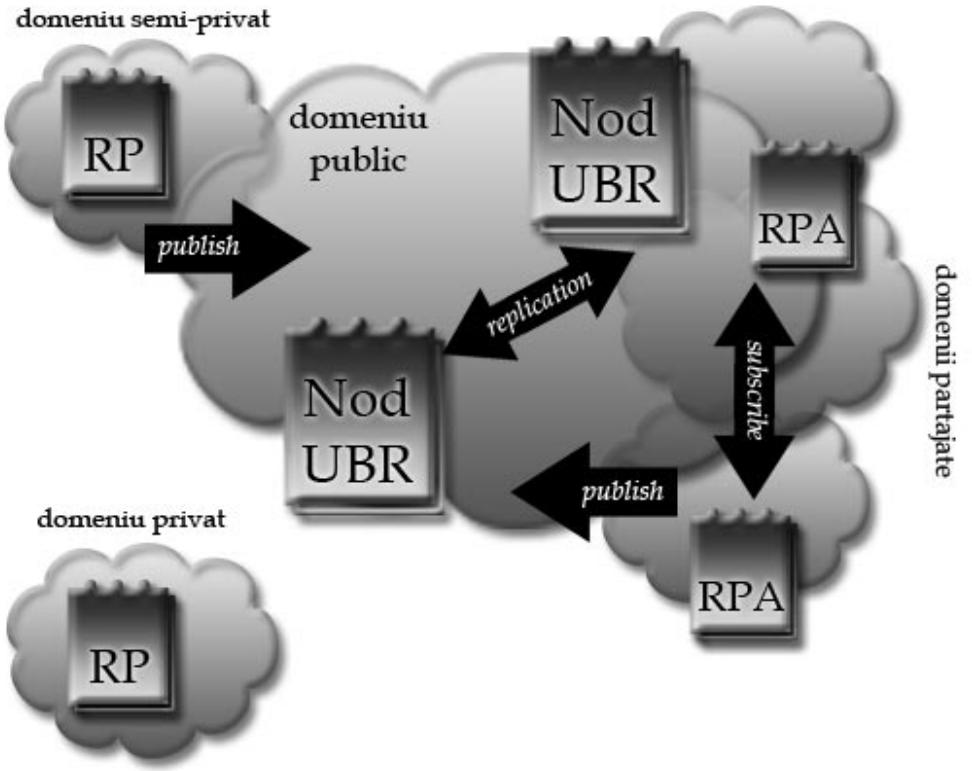


Figura 3. *Tipurile de regiștri și interacțiunile între aceștia (RP = registru privat, RPA = registru privat afiliat, UBR* = UDDI Business Registry*)*

În figura de mai sus sunt descrise o serie de moduri de interacțiune între regiștri. Prin mecanisme de tip publicare/abonare (*publish/subscribe*) și replicare (*replication*) între nodurile unui registru, informațiile de pe serverele UDDI pot fi în totalitate publice (cazul UBR), semi-private sau total private și izolate (situația unei corporații folosind servicii Web interne, la nivel de intranet propriu).

1.6. *Interfețe de programare pentru UDDI*

În cadrul specificațiilor UDDI sunt definite două categorii de interfețe de programare principale: *Publisher API* și *Inquire API*.

Publisher API

Interfața *Publisher API* permite publicarea și actualizarea de informații conținute în regiștrii UDDI. Funcțiile puse la dispoziție sunt descrise de tabelul următor.

Metoda	Descriere
add_publisherAssertions	Adaugă una sau mai multe relaţii specificate de *publisherAssertion*
delete_binding	Şterge una/mai multe instanţe ale elementului *bindingTemplate* din regiştri
delete_business	Şterge una/mai multe înregistrări de afaceri
delete_publisherAssertions	Elimină unul/mai multe elemente *publisherAssertion*
delete_service	Şterge unul/mai multe elemente *businessService*
delete_tModel	Şterge logic una/mai multe structuri *tModel*
discard_authToken	Informează un nod că informaţiile de autentificare vor fi înlăturate
get_assertionStatusReport	Raportează starea declaraţiilor care implică orice înregistrare de afacere a cărui management este realizat de un anumit *publisher*
get_authToken	Obţine un jeton de autentificare
get_publisherAssertions	Furnizează toate relaţiile de tip *publisherAssertion* realizate de un *publisher*
get_registeredInfo	Obţine o listă a tuturor structurilor *businessEntity* şi *tModel*
save_binding	Salvează/actualizează un element *bindingTemplate*
save_business	Salvează/actualizează o structură *businessEntity*
save_service	Adaugă/actualizează unul ori mai multe elemente *businessService*
save_tModel	Salvează sau actualizează unul/mai multe elemente de tip *tModel*
set_publisherAssertions	Înlocuieşte toate declaraţiile de tip *publisherAssertion* ale unui *publisher*

Inquiry API

Interfaţa de programare *Inquiry API* permite operaţii pentru căutarea şi parcurgerea regiştrilor cu scopul de a obţine informaţii despre o anumită afacere sau un serviciu dorit. Tabelul de mai jos sintetizează lista funcţionalităţilor oferite.

Metoda	Descriere
find_binding	Localizează o informaţie de legare (*binding*) a unui serviciu la un protocol (tip, port etc.) în cadrul unui sau mai multor elemente *businessService*
find_business	Localizează informaţii despre una/mai multe afaceri
find_relatedBusinesses	Localizează informaţii despre înregistrările *businessEntity* asociate unei anumite entităţi de afaceri
find_service	Localizează anumite servicii în cadrul unor entităţi de afaceri

Metoda	Descriere
find_tModel	Localizează una/mai multe structuri *tModel*
get_bindingDetail	Obţine informaţii detaliate pentru realizarea de cereri privitoare la *bindingTemplate*
get_businessDetail	Obţine informaţii despre *businessEntity* pentru una sau mai multe afaceri/organizaţii
get_businessDetailExt	Furnizează informaţii detaliate despre *businessEntity*
get_serviceDetail	Obţine detalii complete despre un set de entităţi *businessService*
get_tModelDetail	Furnizează detalii referitoare la un *tModel*

Publisher API şi *Inquiry API* reprezintă nucleul instrumentelor care realizează managementul datelor din cadrul regiştrilor UDDI.

De notat faptul că marile companii pun la dispoziţie cataloage de tip UBR ce pot fi accesate via API-urile prezentate mai sus. Câteva exemple notabile sunt următoarele:

– accesarea interfeţei de interogare a registrului UBR de la IBM: *http://uddi.ibm.com/ubr/inquiryapi*
– accesarea interfeţei de publicare în registrul UBR oferit de IBM: *https://uddi.ibm.com/ubr/publishapi*
– accesarea interfeţei de interogare a registrului UBR pus la dispoziţie de Microsoft: *http://uddi.microsoft.com/inquire*
– accesarea interfeţei de publicare pentru registrul de tip UBR de la Microsoft: *https://uddi.microsoft.com/publish*

Sunt disponibile şi o serie de instrumente grafice de acces la UDDI prin interfeţele prezentate mai sus. Menţionăm doar *Registry Browser* din cadrul pachetului de dezvoltare a serviciilor Web oferit de Sun – Java WSDP (*Web Services Developer Pack*). Tot în cadrul acestui pachet sunt disponibile o serie de exemple de programe Java pentru lucrul cu regiştrii UDDI publici.

Alte interfeţe de programare

O interfaţă de programare suplimentară este *Security API*, oferind două funcţii:

– *discard_authToken* – informează nodul că *token*-ul (jetonul) de autentificare obţinut anterior nu mai este solicitat şi că trebuie considerat invalid dacă este utilizat după ce acest mesaj este primit (astfel, se controlează maniera de valabilitate a procesului de autentificare);
– *get_authToken* – solicită de la un nod UDDI un jeton de autentificare sub forma unui element *authInfo*. Un astfel de element poate fi solicitat atunci

când se folosesc funcții din alte interfețe de programare precum *Inquiry*, *Publication*, *Custody and Ownership Transfer* sau *Subscription*.

Interfața de programare numită *Custody and Ownership Transfer API* permite ca orice nod al unui registru să transfere „custodia" unei structuri *businessEntity* sau *tModel* de la un nod la altul. De asemenea, se oferă suport și pentru preluarea de la o entitate la alta a dreptului de proprietate asupra acestor structuri.

Fără a intra în detalii tehnice – pe care le puteți găsi în specificațiile UDDI – să urmărim un scenariu general. Să considerăm că avem la dispoziție un registru UDDI format din nodurile A, B și C. Registrul permite fiecărui nod să-și definească propriile reguli pentru înregistrare, autentificare și autorizare. Dacă o persoană P dorește să se autentifice, va trebui să obțină regulile de politică a accesului ale celor trei noduri. Să considerăm cazul în care P alege nodul A.

Dacă reușește să se autentifice cu succes și să publice o entitate *business* (fie ea desemnată de E_1), atunci P devine proprietar (*owner*) al lui E_1. Se consideră că nodul A este custode (*custodian*) pentru E_1. De asemenea, P poate efectua aceleași operații la nodul B, fiind în măsură să publice o entitate de afaceri E_2. Nu se poate face nici o presupunere că un registru UDDI (sau nodurile sale) este „conștient" că P reprezintă aceeași persoană. P este responsabil pentru menținerea identității și autentificării corecte pentru fiecare nod din cadrul registrului.

Interfața *Subscription API* dă posibilitatea clienților autentificați să primească informații privitoare la schimbările regiștrilor UDDI de care aceștia sunt interesați. Această interfață este opțională și poate fi implementată în maniera dorită la nivelul fiecărui nod.

1.7. *UDDI și SOAP*

Vom enumera în cele ce urmează convențiile utilizării SOAP și UDDI, prezentând ce elemente ale protocolului SOAP 1.1 sunt suportate sau nu de UDDI:

- UDDI necesită prezența câmpului *SOAPAction* din antetul HTTP. *SOAPAction* poate conține o valoare sau poate fi vid:

```
POST ... HTTP/1.1
Host: www.portocale.info
Content-Type: text/xml; charset="utf-8"
Content-Length: NNNN
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<Body>
   <get_bindingDetail xmlns="urn:uddi-org:api_v3">
...
</Envelope>
```

- UDDI nu oferă suport pentru conceptul de actor SOAP (pentru SOAP 1.2, regăsit sub numele de rol – *role*). Nodurile UDDI vor respinge orice cerere care vine cu un atribut *actor* în elementul *Header*. Se va întoarce o eroare SOAP fără nici un element *Detail*, iar codul erorii este de tip *Client* (sau *Sender* în cazul SOAP 1.2).
- UDDI nu are prevăzut suport pentru facilitatea *SOAP Encoding*. În mesajele trimise unui nod UDDI, nu trebuie să existe nici o informație privitoare la stilul de codificare pentru nici un element din cadrul spațiului de nume *urn:uddi-org*. În caz contrar, nodul va răspunde cu un cod de eroare de tip *Client*, iar *faultstring* va indica motivul apariției problemei.
- Regiștrii UDDI pot ignora conținutul anteturilor SOAP.
- UDDI suportă următoarele coduri de eroare SOAP 1.1: *VersionMismatch, MustUnderstand*, *Client* și *Server*.

1.8. *UDDI și WSDL*

UDDI este o arhitectură care permite stocarea informațiilor privitoare la diferite servicii Web, însă nu se ocupă cu descrierea acestora. Pentru a putea utiliza un serviciu Web regăsit prin intermediul UDDI, avem nevoie de o descriere WSDL a acestuia. În cele ce urmează, ne vom referi la WSDL 1.1, modificările în cele ce privește versiunea 2.0 fiind minore.

Specificațiile existente descriu câteva tipuri de structuri *tModel*, care trebuie să fie furnizate de regiștrii UDDI, astfel încât să putem avea suport pentru a modela informații de tip *portType* și *binding* din WSDL la nivel UDDI. De asemenea, trebuie furnizat un model care să fie consistent atunci când se lucrează atât cu versiunea 3.0, cât și cu UDDI 2.0. Mai mult, surprinderea informațiilor dintr-o WSDL trebuie să se realizeze fără o duplicare excesivă.

Sunt introduse șase așa-numite sisteme categoriale (*category systems*):

1. *WSDL Entity Type Category System* – este folosit pentru a identifica entitățile UDDI care corespund celor WSDL. Este în mod obișnuit folosit în cazul structurilor *tModel* și *businessService*. Cum WSDL permite ca *portType* și un *binding* să aibă același nume, singurul mod de a diferenția cele două elemente *tModel* – corespunzătoare informațiilor *portType* și, respectiv, *binding* – este de a recurge la *keyedReference*.

2. *XML Namespace Category System* – este utilizat la reprezentarea oricărui spațiu de nume XML (nu doar pentru spații de nume privitoare la WSDL). Această facilitate permite realizarea de interogări precise, folosindu-se numele complet al elementelor XML (spațiu de nume + numele local de element).
3. *XML Local Name Category System* – poate fi util atunci când se folosește numele local al unui element XML, pentru cazul în care numele corespunzând unei structuri UDDI nu poate fi același cu cel al unui element XML local (fără a fi prefixat de spațiul de nume).
4. *WSDL portType Reference Category System* – este folosit pentru a reprezenta legătura dintre un *binding* și un *portType*. I se permite unei entități UDDI să se refere la o altă entitate prin stocarea cheii sale ca valoare a atributului *keyValue* al lui *keyReference* (privitor la acest sistem de clasificare).
5. *Protocol Category System* – permite ca structuri *tModel* referitoare la protocoale definite de utilizator să fie încorporate în abordarea descrisă în specificații.
6. *Transport Category System* – permite ca structuri *tModel* de transport specificate de utilizator să poate fi folosite în acest context.

De asemenea, se mai poate recurge și la următoarele tipuri de structuri *tModel*:

– *SOAP Protocol tModel* – pentru a reprezenta utilizarea SOAP și WSDL;
– *HTTP Protocol tModel* – pentru a modela legătura între WSDL și HTTP;
– *WSDL Address tModel* – pentru cazul în care se folosesc documente WSDL.

Vom da o serie de detalii referitoare la legătura dintre WSDL și UDDI 2.0 (diferențele față de versiunea 3 nu sunt semnificative).

Un element *portType* specific WSDL este reprezentat în cadrul UDDI de un *tModel*. Acest *tModel* este clasificat folosind *WSDL Entity Type Category System*. Numele structurii *tModel* asociate lui *portType* va fi identic cu cel al lui *portType*. Acest aspect eludează cumva specificațiile UDDI care prevăd că numele unui *tModel* ar trebui să fie un URI.

Când s-a luat această decizie s-au propus următoarele alternative:

– să se folosească numele lui *portType* – varianta aleasă;
– să se recurgă la o combinație dintre *portType* și spațiul de nume. Această opțiune ar fi făcut mai dificilă căutarea fie după spațiul de nume, fie conform unui *portType*;
– să se utilizeze un URI și numele local al elementelor XML, iar spațiul de nume să fie plasat în sub-elementul *categoryBag* din *tModel*. Această variantă face și mai dificilă căutarea numelui unui *portType*, iar numele *tModel*-ului nu conține nici o informație reprezentativă.

Spațiul de nume al elementului *portType* este dat de *keyedReference* din marcatorul *categoryBag* al structurii *tModel* privitoare la *portType*. Acest *keyedReference* determină legătura cu sistemul de clasificare *XML Namespace* menționat mai sus.

Elementul *binding* este reprezentat tot ca un *tModel*, dar de tip *WSDL binding tModel*, pentru a-l distinge de alte categorii de structuri *tModel*. Se folosește același sistem de clasificare ca și în cazul anterior, dar cu o valoare diferită pentru *keyValue*. Numele *tModel*-ului este același cu cel cuprins de *binding*.

Elementul *service* al documentului WSDL este reprezentat de *businessService*. Dacă *service* desemnează interfața unui serviciu Web prezent deja în cadrul registrului UDDI, atunci structura *businesService* aferentă este adăugată informațiilor existente. În caz contrar, poate fi creată o nouă intrare *businessService*.

Elementul *port* este reprezentat de *bindingTemplate*. Conținutul relației dintre un serviciu WSDL și porturile sale este oglindit aici de relația dintre *businessService* și *bindingTemplates*.

Discuția de mai sus se concretizează în figura următoare.

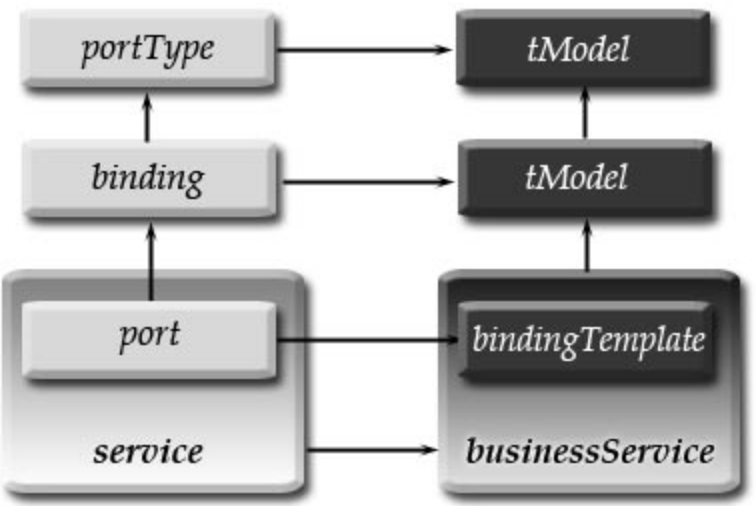


Figura 4. *Relația dintre elementele WSDL și cele UDDI*

Publicarea și găsirea descrierilor WSDL

O descriere WSDL completă a unui serviciu este o combinație între documentul de tip interfață a serviciului și documentul de tip implementare.

Cum interfața serviciului reprezintă o definiție reutilizabilă a acestuia, ea este publicată într-un registru UDDI ca un *tModel*. Implementarea serviciului descrie o instanță a acestuia. Fiecare instanță este definită folosind elementul WSDL

service. Pentru a publica o structură *businessService* este utilizat un element *service* dintr-un document de tip implementare.

Figura 5 oferă o serie de detalii privind asocierea elementelor WSDL ca elemente UDDI.

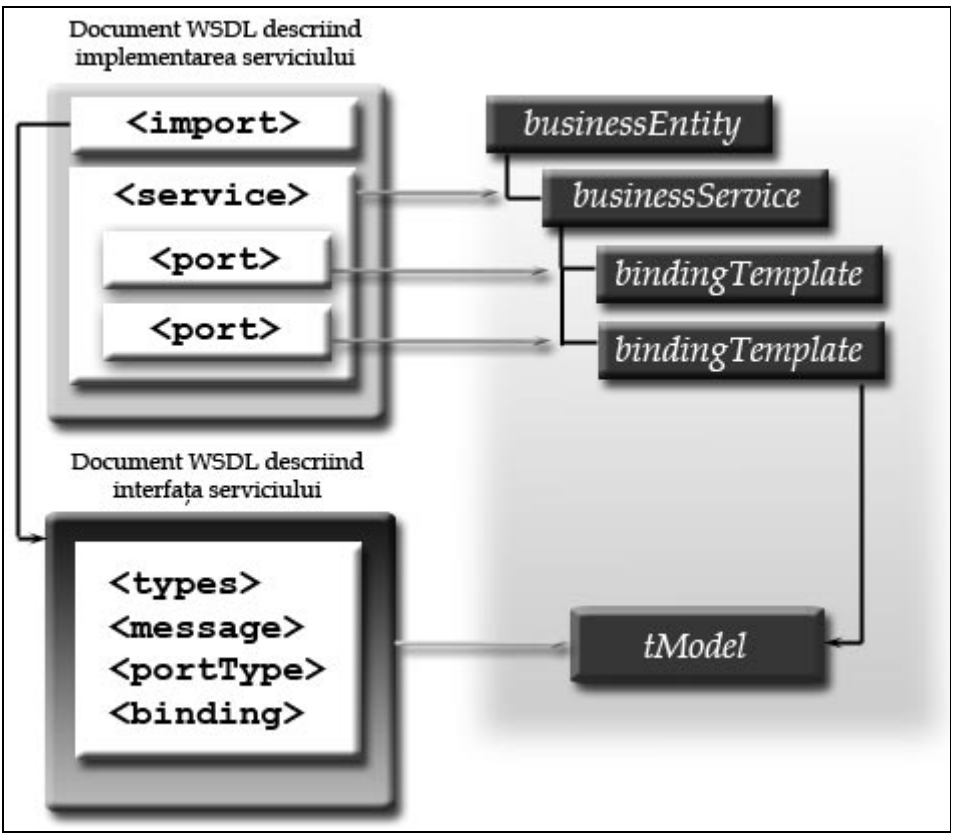


Figura 5. *Corespondența dintre elementele WSDL și intrările UDDI*

Structura *tModel* corespunzătoare interfeței serviciului Web este publicată de furnizorul acelei interfețe de serviciu. O parte dintre elementele din *tModel* sunt construite pe baza informațiilor din descrierea WSDL aferentă. În tabelul următor se definesc pașii creării unui *tModel* care pentru a fi valid ar trebui numit folosind *targetNamespace* și ar trebui să conțină date pentru „umplerea" elementelor *overviewURL* și *categoryBag*.

	tModel UDDI	Interfața WSDL a serviciului	Descriere	Obligatoriu
1	*name*	Atributul *targetNamespace* pentru definirea elementului	Numele pentru *tModel* este obținut folosindu-se valoarea atributului *targetNamespace* din documentul WSDL. Este necesar un nume consistent, pentru ca un *tModel* să fie localizat folosindu-se doar informații din documentul de tip implementare a serviciului.	Da
2	*description*	Elementul *documentation*	Elementul *description* din *tModel* este restrâns la 256 de caractere. Dacă elementul *documentation* nu există, atunci trebuie utilizat numele atributului din elementul *definitions*.	Nu
3	*overviewURL*	URI-ul interfeței serviciului și specificarea elementului *binding*	Localizarea documentului de tip interfață a serviciului poate fi folosită ca valoare pentru *overviewURL*. Dacă există mai mult de un singur element *binding*, atunci valoarea acestuia este codificată în URL.	Da
4	*categoryBag*	–	Elementul *categoryBag* trebuie să conțină cel puțin un sub-element *keyedReference*. Atributul *keyName* va avea valoarea *uddi-org:types*, iar *keyValue* va avea valoarea *wsdlSpec*.	Da

Documentul de mai jos reprezintă interfața unui serviciu Web. Valorile ce vor fi inserate într-un *tModel* apar cu caractere îngroșate:

```
<?xml version="1.0"?>
<definitions name="TranzactiiPortService-interface"
  targetNamespace=
    "http://www.portocale.info/
     TranzactiiPortService-interface"
  xmlns:tns=
   "http://www.portocale.info/TranzactiiPortService-interface"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <documentation xml:lang="ro">
   Definiția unei interfețe standard WSDL pentru
   un serviciu de tranzacții de portocale.
  </documentation>
```

```
  <message name="TipPortocaleRequest">
    <part name="tip" type="xsd:string"/>
  </message>
  <message name="TipPortocaleResponse">
    <part name="tipIntors" type="xsd:string"/>
  </message>
  <portType name="TipPortocaleTranzactiiPortService">
    <operation name="furnizeazaTipPortocale">
      <input message="tns:TipPortocaleRequest"/>
      <output message="tns:TipPortocaleResponse"/>
    </operation>
  </portType>

  <binding name="TipPortocaleBinding"
    type="tns:TipPortocaleTranzactiiPortService">
    <soap:binding style="rpc" transport=
      "http://schemas.xmlsoap.org/soap/http"/>
      <operation name="furnizeazaTipPortocale">
          <soap:operation soapAction=
             "http://www.portocale.info/furnizeazaTip"/>
      <input>
        <soap:body use="encoded"
          namespace="urn:tip-portocale-tranzactii-port"
          encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
          namespace="urn:tip-portocale-tranzactii-port"
          encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>

</definitions>
```

Valoarea atributului *targetNamespace* va fi folosită ca nume pentru *tModel*, conținutul elementului *documentation* va reprezenta descrierea structurii *tModel*, iar atributul *name* al elementului *binding* va fi utilizat să descrie *overviewURL*.

Documentul UDDI privitor la *tModel* are forma următoare:

```
<?xml version="1.0"?>
<tModel tModelKey="">
  <name>http://www.portocale.info/
    TranzactiiPortService interface</name>
```

```
  <description xml:lang="ro">
   Definiția unei interfețe standard WSDL pentru
   un serviciu de tranzacții de portocale.
  </description>

  <overviewDoc>
    <description xml:lang="ro">
      Documentul WSDL descriind interfața serviciului
    </description>
    <overviewURL>
     http://www.portocale.info/TranzactiiPortService-
       interface.wsdl#TipPortocaleBinding
    </overviewURL>
  </overviewDoc>

  <categoryBag>
    <keyedReference tModelKey="uuid:..."
        keyName="uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference tModelKey="uuid:..."
        keyName="Serviciu tranzact." keyValue="..."/>
  </categoryBag>
</tModel>
```

Elementul *overviewURL* are valoarea *http://www.portocale.info/TranzactiiPortService-interface.wsdl*, care reprezintă adresa documentului WSDL de tip interfață a serviciului Web. El conține, de asemenea, o referință directă la elementul *binding* cu `name="TipPortocaleBinding"` în documentul de tip interfață a serviciului. Cum există doar un singur element *binding* în documentul WSDL, această referință la *binding* nu este obligatorie.

Implementarea serviciului Web este publicată în regiștrii UDDI ca o structură *businessService*, având unul sau mai multe marcaje *bindingTemplates*. Această publicare este realizată de furnizorul acelui serviciu.

Pentru fiecare element *service* specificat în documentul de implementare a serviciului va fi creată câte o structură *businessService*.

Lista de sub-elemente ale *businessService* care pot fi specificate pe baza conținutului documentului WSDL de implementare a serviciului cuprinde:

– *name* – are valoarea dată de atributul elementului *service* din documentul WSDL;
– *description* – folosește primele 256 de caractere ale elementului *documentation*, dacă există.

Pentru fiecare *port* din cadrul elementului *service* se creează un nou element *bindingTemplate*, conform detaliilor din tabelul următor.

	binding Template	**Descriere**	**Obligatoriu**
1	*description*	Dacă elementul *port* conţine un sub-element *documentation*, descrierea conţine primele 256 de caractere ale elementului *documentation*	Nu
2	*accessPoint*	Pentru o legare (*binding*) SOAP sau HTTP, *accessPoint* are valoarea atributului *location* al elementului *extension* asociat elementului *port*. Dacă elementul conţine un URL, atributul *URLType* va desemna protocolul din cadrul URL-ului. Dacă nu se foloseşte nici un URL, atributul *URLType* ar trebui folosit pentru a se cunoaşte tipul protocolului.	Da
3	*tModel InstanceInfo*	Va exista câte un element *tModelInstanceInfo* pentru fiecare *tModel* pe care-l referenţiază. Astfel, va apărea cel puţin un *tModelInstanceInfo* incluzând o referinţă directă la *tModel*-ul ce reprezintă documentul de interfaţă a serviciului.	Da
4	*overviewURL*	Poate conţine o referinţă directă la documentul de implementare a serviciului, folosită doar pentru a furniza o documentaţie uşor de înţeles (*human-readable*). Toate celelalte informaţii din acest document ar trebui să fie accesibile printr-o entitate de date UDDI (*UDDI data entity*). Suntem asiguraţi că documentul publicat este acelaşi cu cel obţinut în urma operaţiei de căutare. Dacă acest document cuprinde mai mult decât un element *port*, atunci acesta ar trebui să conţină o referinţă directă la atributul *name* al lui *port*. Nu este suficient să se folosească doar o referinţă la elementul *binding* din *tModel*, deoarece pot exista mai multe marcaje *port* referind acelaşi *binding*.	Nu

Exemplul de mai jos reprezintă un document WSDL de implementare a unui serviciu. Valorile indicate cu caractere îngroşate sunt obligatorii pentru crearea intrărilor *businessService* şi *bindingTemplates*.

```
<definitions name="TranzactiiPortService"
  targetNamespace=
    "http://www.portocale.info/TranzactiiPortService"
  xmlns:interface=
    "http://www.portocale.info/
       TranzactiiPortService-interface"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
<import namespace="http://www.portocale.info/
  TranzactiiPortService-interface"
  location=
    "http://www.portocale.info/wsdl/
       TranzactiiPortService-interface.wsdl"/>

<service name="TranzactiiPortService">
  <documentation>...</documentation>
  <port name="TipPortocaleServicePort"
        binding="interface:TipPortocaleBinding">
     <documentation>
        TipPortocaleTranzactiiPortService
     </documentation>
     <soap:address location=
     "http://www.portocale.info/TranzactiiPortService"/>
  </port>
</service>
</definitions>
```

Atributul *name* al elementului *service* va fi folosit ca nume pentru *businessService*. Elementul *documentation* din interiorul elementului *service* este utilizat pentru a descrie structura *businessService*.

Atributul *name* al elementului *port* este anexat lui *overviewURL*, care conține o referință la documentul de implementare a serviciului. Atributul *location* al lui *service* este folosit pentru „umplerea" marcatorului *accessPoint* din structura *bindingTemplate*.

Definiția unei intrări *businessDefinition* create pe baza documentului WSDL de mai sus este furnizată de exemplul următor. Elementul *categoryBag* ar trebui să conțină unul sau mai mulți marcatori *keyedReference*, folosiți în clasificarea scopurilor afacerilor utilizate pentru serviciul de față.

```
<businessService businessKey="..." serviceKey="...">
  <name>TranzactiiPortService</name>
  <description xml:lang="en">
     ...
  </description>

  <bindingTemplates>
    <bindingTemplate bindingKey="..." serviceKey="...">
      <description>
         ...
      </description>
      <accesssPoint URLType="http">
```

```
          http://www.portocale.info/TipTranzactiiPort
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="...">
            <instanceDetails>
              <overviewURL>
               http://www.portocale.info/services/
                     TranzactiiPortService.wsdl
              </overviewURL>
            </instanceDetails>
        </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>

  <categoryBag>
    <keyedReference tModelKey="uuid:..."
        keyName="..." keyValue="..." />
  </categoryBag>
</businessService>
```

Atributul *tModelKey* are drept valoare un UUID pentru *tModel*-ul privitor la documentul de interfață a serviciului. Acest *tModel* poate fi localizat folosindu-se atributul *namespace* din cadrul elementului *import*. Marcatorul *overviewURL* reprezintă locația documentului care implementează serviciul. El nu conține o referință la elementul *port*, deoarece este unic pentru acest document.

Găsirea descrierilor WSDL privitoare la interfața serviciilor se realizează după cum urmează. După cum deja cunoaștem, toate documentele WSDL referitoare la interfața serviciilor Web sunt publicate în regiștrii UDDI sub formă de *tModel*. Fiecare structură *tModel* e clasificată într-o anumită manieră via *categoryBag*. Operația *find_tModel* oferită de API-ul UDDI poate fi folosită pentru a găsi un *tModel* clasificat. Localizarea tuturor descrierilor WSDL privitoare la interfața unui serviciu se realizează conform construcției următoare:

```
<find_tModel generic="2.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference tModelKey="uuid:..."
                    keyName="uddi-org:types"
                    keyValue="wsdlSpec" />
  </categoryBag>
</find_tModel>
```

Operația *find_tModel* va întoarce o listă de chei *tModel*. Folosindu-se metoda *get_tModelDetail*, este obținută o descriere particulară a interfeței unui serviciu. Metoda *get_tModelDetail* va returna un element *tModel* precum cel prezentat în cadrul capitolului de față.

Următorul pas este să se recurgă la elementul *overviewURL* pentru a se obține conținutul documentului WSDL de interfață a serviciului.

În plus, în cadrul *categoryBag* se mai pot adăuga elemente *keyedReference* suplimentare pentru a se limita mulțimea de structuri *tModel* întoarse ca răspuns la această cerere. Metoda *find_tModel* de mai sus poate fi folosită pentru a localiza toate serviciile de tip „TranzactiiPort" care folosesc WSDL.

```
<find_tModel generic="2.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference tModelKey="uuid:..."
                    keyName="uddi-org:types"
                    keyValue="wsdlSpec" />
    <!-- inserăm un criteriu suplimentar de căutare -->
    <keyedReference tModelKey="uuid:..."
                    keyName="TranzactiiPort"
                    keyValue="..." />
  </categoryBag>
</find_tModel>
```

Găsirea descrierilor WSDL privitoare la implementarea serviciilor Web se efectuează conform etapelor descrise în continuare. O implementare a unui serviciu este publicată în regiștrii UDDI ca o intrare *businessService*. Acest *businessService* va conține unul sau mai multe elemente *bindingTemplates*.

Suplimentar, structurile *businessService* au asociate categorii de clasificare, atfel încât să poată fi identificate ca descrieri bazate pe WSDL a serviciilor.

O soluție pentru regăsirea descrierii unui serviciu este dată de operația *find_service*.

O anumită entitate *businessEntity* se poate folosi de mesajul următor pentru localizarea unei structuri de tip *businessService*, incluzând implementarea unui serviciu „TranzactiiPort". În plus, pot fi adăugate și alte elemente *keyedReferences* în cadrul *categoryBag*, cu rol de filtrare a răspunsului.

```
<find_service businessKey="..." generic="2.0"
              xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference tModelKey="uuid:..."
                    keyName="TranzactiiPort"
                    keyValue="..." />
  </categoryBag>
</find_service>
```

Metoda *find_service* poate fi de ajutor și pentru a localiza date de tip *businessService*, desemnând implementări specifice ale interfeței unui serviciu. Mesajul descris mai jos conține un exemplu în care se caută toate elementele *businessService* din

cadrul marcajului *businessEntity*. Acest din urmă element referă implementarea interfeței unui serviciu particular (de exemplu, privitor la tranzacțiile bancare). Cum interfața unui serviciu este reprezentată de un *tModel*, se va folosi un *tModelBag* pentru a menționa cheia *tModel* corespunzătoare interfeței WSDL a serviciului considerat.

```
<find_service businessKey="..." generic="2.0"
              xmlns="urn:uddi-org:api">
  <tModelBag>
    <!-- cheia tModel-ului pentru interfața
         serviciului Web -->
    <tModelKey>...</tModelKey>
  </tModelBag>
</find_service>
```

Metoda *find_service* va returna o listă de intrări. Descrierea *businessService* poate fi obținută folosind metoda *get_serviceDetail*. Aceasta va întoarce o structură *businessService* precum cea prezentată mai sus. După ce s-a obținut un *businessService*, pentru invocarea serviciului Web găsit se va selecta un anumit element *bindingTemplate* inclus în *businessService*. Elementul *accessPoint* din cadrul marcatorului *bindingTemplate* semnifică tocmai punctul terminal al serviciului.

De asemenea, poate fi folosit elementul *overviewURL*, în vederea obținerii conținutului documentului WSDL referitor la implementarea serviciului, care la rândul său poate oferi și alte detalii de interes.

Furnizăm în continuare maniera de găsire a unei structuri *bindingTemplate*. Dacă marcatorul *businessService* are mai mult de un element *bindingTemplate*, este dificil să se determine care marcaj *bindingTemplate* trebuie efectiv folosit. Pentru a localiza elementul *bindingTemplate* ce ar trebui utilizat, vom recurge la operația *find_binding*.

Exemplul de mai jos recurge la *find_binding* pentru a obține elementul *bindingTemplate*, referențiind un anumit *tModel*. Acest *tModel* poate fi asociat unui element *binding* particular din cadrul descrierii WSDL a interfeței serviciului.

```
<find_binding serviceKey="..." generic="1.0"
              xmlns="urn:uddi-org:api">
  <tModelBag>
    <!-- cheia corespunzătoare
         interfeței serviciului -->
    <tModelKey>...</tModelKey>
  </tModelBag>
</find_binding>
```

Se va obține unul sau mai multe elemente *bindingTemplate*. După accesarea marcajului *bindingTemplate*, punctul de destinație pentru serviciul Web este oferit

de elementul *accessPoint*. Dacă elementul *bindingTemplate* a fost creat pe baza unui document WSDL existent (privitor la implementarea serviciului), atunci *overviewURL* poate oferi o referință la acest document. Acest document WSDL poate fi accesat pentru a obține și alte informații utile privitoare la serviciul Web, informații care nu au putut fi regăsite în regiștrii UDDI.

Iată un exemplu de structură *bindingTemplate* întoarsă în urma interogării descrise anterior:

```
<bindingTemplate bindingKey=""serviceKey="">
  <accesssPoint URLType="http">
     http://www.portocale.info/TipTranzactiiPort
  </accessPoint>
  <tModelInstanceDetails>
       <!-- tModelKey include o cheie corespunzătoare
            interfeței serviciului -->
    <tModelInstanceInfo tModelKey="...">
      <instanceDetails>
        <overviewURL>
http://www.portocale.info/TranzactiiPortService.wsdl
        </overviewURL>
      </instanceDetails>
    </tModelInstanceInfo>
  </tModelInstanceDetails>
</bindingTemplate>
```

1.9. *Crearea propriului registru UDDI folosind* jUDDI

După cum am văzut mai sus, UDDI definește un protocol de descoperire a serviciilor Web (permițând clienților să regăsească serviciile Web dorite), punând la dispoziție și un format standard de descriere a serviciilor (care oferă clienților detaliile privitoare la funcționalitățile serviciilor Web găsite).

Există două tipuri de regiștri UDDI: regiștrii publici și cei privați. În această secțiune vom detalia în principal maniera de manipulare a regiștrilor privați ce pot fi utili în cadrul unui mediu de tip *enterprise*. De asemenea, vom oferi o serie de amănunte privitoare la un cadru de lucru folosit pentru a reprezenta structuri ale regiștrilor UDDI și pentru a implementa regiștri UDDI privați. Este vorba de *framework*-ul *jUDDI,* disponibil în regim *open source*.

Descriere succintă a instrumentului jUDDI

Instrumentul *jUDDI* este implementat în Java și oferă compatibilitatea cu UDDI versiunea 2.0 pentru managementul regiștrilor UDDI.

Pentru a putea crea propriul registru UDDI, trebuie instalate următoarele aplicații:

- *Java 2 SDK* – în exemplificările următoare am folosit Java 2 SDK SE, versiunea 1.5.0_06;
- *J2EE Build and Runtime Environment* – s-a utilizat *Java 2 Enterprise Edition* (J2EE) 1.4 oferit de Sun;
- Un server Web și/sau un container de *servlet*-uri – în acest caz s-a recurs la *Apache Tomcat*, versiunea 5.5.12;
- Un cadru de lucru pentru procesarea mesajelor SOAP – s-a folosit *Apache Axis* (compatibil cu *jUDDI*).
- Un mecanism de stocare a datelor – s-a utilizat serverul de baze de date MySQL versiunea 5.0 (se poate recurge, de asemenea, și la alte soluții, eventual comerciale: DB2, Sybase etc.);
- Un instrument de management al regiștrilor UDDI – în cazul nostru, am adoptat *jUDDI*.

Aplicația *jUDDI* folosește *Apache Axis* pentru manipularea mesajelor SOAP. *Axis* definește un cadru de transport transparent, care permite folosirea diferitelor protocoale. Pentru transferul mesajelor via HTTP, orice *servlet* derivat din clasa *org.apache.axis.transport.http.AxisServlet* este un bun candidat.

În *jUDDI* există disponibile trei *servlet*-uri ce extind clasa *AxisServlet*:

– *org.apache.juddi.transport.axis.AdminServlet* – pentru administrare;
– *org.apache.juddi.transport.axis.PublishServlet* – pentru publicarea în regiștri;
– *org.apache.juddi.transport.axis.InquiryServlet* – pentru realizarea interogărilor.

Aceste trei clase sunt înregistrate de *jUDDI* ca *servlet*-uri cu ajutorul unei aplicații--server și sunt folosite doar pentru a determina tipul cererilor care sunt efectuate. Procesarea de fapt este realizată de către clasa *org.apache.juddi.transport.axis.AxisHandler*.

Instrumentul jUDDI încapsulează structurile de date fundamentale UDDI (*businessEntity*, *businessService*, *bindingTemplate* și *tModel*) în clase. Acestea se găsesc în pachetul *org.apache.juddi.datatype* sub următoarele denumiri:

– *org.apache.juddi.datatype.business.BusinessEntity*;
– *org.apache.juddi.datatype.service.BusinessService*;
– *org.apache.juddi.datatype.binding.BindingTemplate*;
– *org.apache.juddi.datatype.tmodel.TModel.*

Pentru a putea procesa cererile clienților, nu avem de făcut decât să recurgem la instanțele acestor clase, împreună cu alte tipuri de date furnizate de mediul *jUDDI*.

Manipularea cererilor via jUDDI

jUDDI despachetează un mesaj de cerere cu ajutorul unui obiect numit *AxisHandler*. Clasa *AxisHandler* folosește serviciile clasei *RegistryEngine* pentru a efectua procesarea efectivă a cererii primite. Modul de departajare a tipurilor de cereri (*inquiry*, *publish* și *admin*) se realizează prin intermediul unui URI în cadrul *AxisServlet. Framework*-ul clasifică fiecare cerere în raport cu valoarea proprietății găsită în obiectul de cerere *MessageContext* pentru cheia *transport.http.servlet*.

De exemplu, o cerere de publicare de date în regiștrii UDDI e desemnată de un URI de genul *http://localhost:8080/juddi/publish* asociat *servlet*-ului responsabil cu aceasta: *PublishServlet*. După ce a primit o cerere, un obiect al clasei *RegistryEngine* convertește cererea UDDI bazată pe XML în obiecte Java (deserializare, *unmarshalling*). Se invocă obiectele Java și ulterior se convertește răspunsul (obiectele Java) într-un răspuns în format XML (serializare, *marshalling*).

Similar, un URI de forma *http://localhost:8080/juddi/inquiry* corespunde unui *servlet* de tip *InquiryServlet*.

Modul de autentificare

Framework-ul *jUDDI* recurge la *org.apache.juddi.auth.AuthenticationFactory* pentru a crea o instanță *Authenticator*, cu scopul de a putea autentifica un client dat. *AuthenticatorFactory* este o implementare a modelului *Factory* și se folosește pentru a realiza o implementare pentru interfața *org.apache.juddi.auth.Authenticator*. Dacă se folosește o valoare nulă, atunci *AuthenticationFactory* creează o implementare implicită pentru *Authenticator – org.apache.juddi.auth.DefaultAuthenticator*. Clasa *DefaultAuthenticator* nu aplică nici o restricție de acces, deci se permit toate cererile.

În general, sistemele construite pe baza *jUDDI* trebuie să ofere o implementare pentru interfața *Authenticator*, care să realizeze procesul de autentificare dorit. Clasa care implementează *Authentication* este înregistrată printr-o intrare a fișierului de configurare *juddi.properties*.

Crearea regiștrilor

Aplicația *jUDDI* se instalează ca o aplicație Web (fișier *.ear* sau *.war*) și, în contextul dat de adresa *http://gazdă:port/juddi*, se poate exploata oriunde există un server Web sau un motor de procesare a *servlet*-urilor (*servlet engine*).

Vom descrie în continuare pașii care trebuie parcurși pentru instalare. Precizăm faptul că, pentru ca instalarea să reușească pe mașina dumneavoastră conform „rețetelor" furnizate mai jos, trebuie să recurgeți exact la versiunile de aplicații precizate. În cazul utilizării altor versiuni, vor fi necesare – eventual – configurări diferite.

Preluați de pe Web serverul *Apache Tomcat* (versiunea 5.5). După ce l-ați instalat, testați funcționarea lui la adresa *http://localhost:8080*.

Transferați apoi de pe situl Apache *framework*-ul *jUDDI*. Pentru exemplificările noastre am utilizat *juddi.war*, versiunea 0.7.0.

Prin intermediul *Tomcat Web Application Manager*, încărcați *juddi.war* (a se vedea figura următoare).

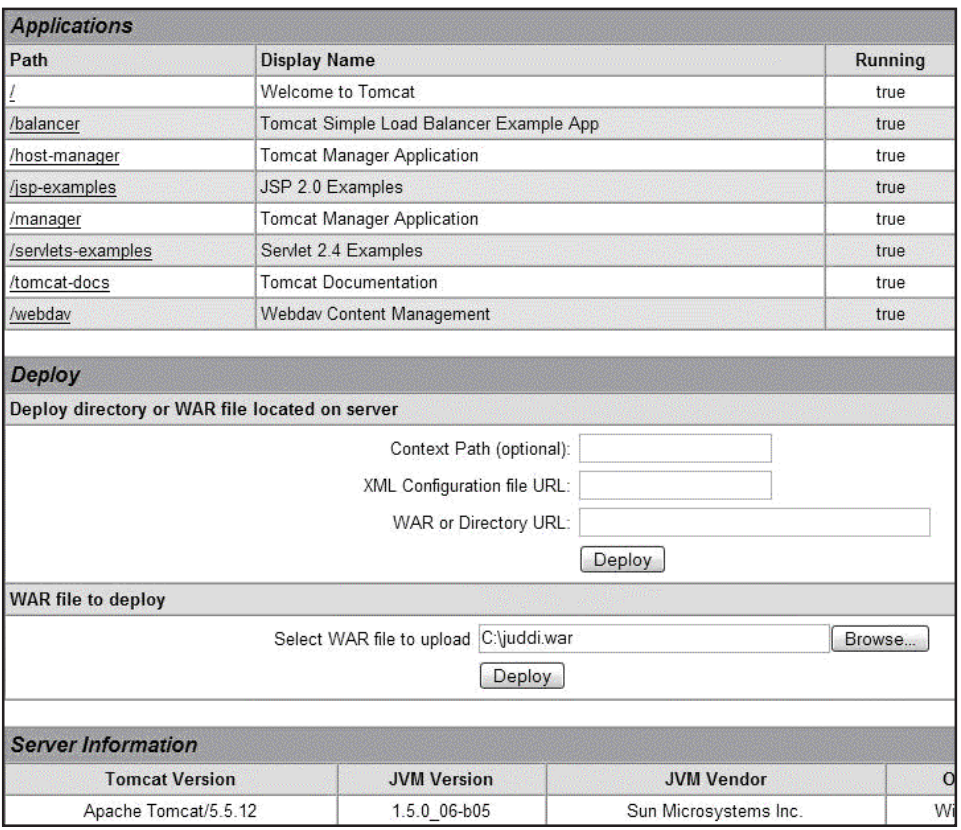


Figura 6. *Interfața Web a programului de management al serverului Tomcat*

În caz de succes, veți obține un rezultat precum cel din captura-ecran de mai jos.

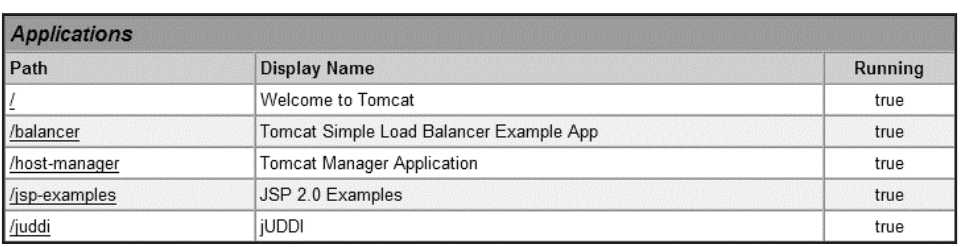


Figura 7. *Inserarea cu succes a* framework*-ului jUDDI*

Pentru a stabili fișierul de jurnalizare corespunzător aplicației *jUDDI*, vom edita *log4j.properties* (localizat, pentru Windows, în directorul *c:\Tomcat5.5\webapps\juddi\conf*), astfel încât să includă și linia:

```
log4j.appender.juddilog.File=C:/Tomcat5.5/logs/juddi.log
```

De asemenea, trebuie modificat și fișierul *c:\Tomcat5.5\webapps\juddi\WEB-INF\web.xml* pentru a conține și următoarele:

```
<env-entry-name>
   log4j.propsFile
</env-entry-name>
<env-entry-value>
   C:\Tomcat5.5\webapps\juddi\conf\log4j.properties
</env-entry-value>

<env-entry-name>
   juddi.propsFile
</env-entry-name>
<env-entry-value>
   C:\Tomcat5.5\webapps\juddi\conf\juddi.properties
</env-entry-value>
```

Următorul pas este să instalați serverul MySQL (am lucrat cu versiunea 5.0.18). Transferați apoi instrumentul *MySQL Control Center*, care este un client folosit pentru a crea și a interoga baza de date.

În acest moment, trebuie să creăm baza de date folosită de *jUDDI*; va fi numită *juddi*. Autentificarea la server se va realiza prin utilizatorul „*root*" cu parola „*adria*". Încărcați fișierul *c:\Tomcat5.5\webapps\juddi\ddl\juddi_mysql.ddl* într-o fereastră SQL și executați comenzile SQL pe care le conține (a se urmări figura de mai jos).

Pasul următor este executarea următoarei comenzi, care stabilește numele entității care va publica date în regiștrii privați UDDI:

```
INSERT INTO PUBLISHER
  (PUBLISHER_ID,PUBLISHER_NAME,ADMIN)
  VALUES ('juddi', 'Juddi user', 'false');
```

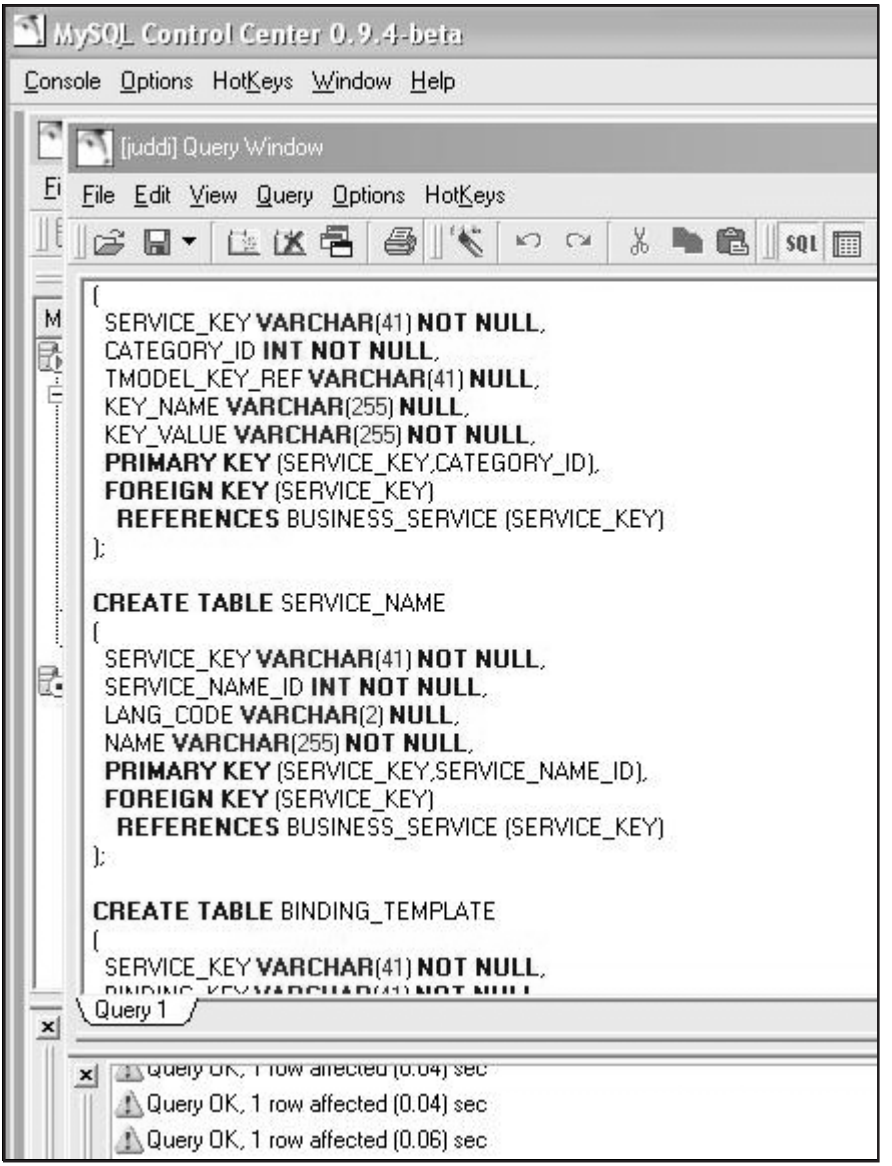


Figura 8. *Execuția fișierului de comenzi SQL în vederea creării tabelelor ce urmează a fi folosite de* jUDDI

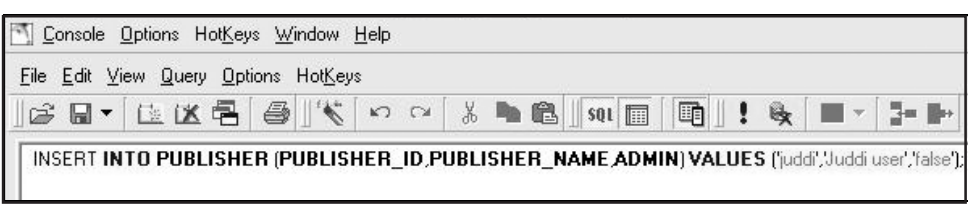

Console Options HotKeys Window Help
File Edit View Query Options HotKeys
SQL
INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN) VALUES ('juddi','Juddi user','false');

Figura 9. *Inserarea datelor privitoare la entitatea ce va publica date în regiștrii privați*

Mai trebuie să obțineți de pe Internet *driver*-ul MySQL JDBC, astfel încât să se poată accesa serverul MySQL. Preluați *mysql-connector-java-3.0.17-stable-bin.jar* și copiați-l în directorul *c:\Tomcat5.5\common\lib*.

Configurarea și folosirea aplicației jUDDI

Creați un fișier cu numele *juddi.xml* în cadrul directorului *c:\Tomcat5.5\conf\Catalina\localhost*. Drept conținut, introduceți următoarele linii:

```
<Context path="/juddi" docBase="juddi" debug="5"
         reloadable="true" crossContext="true">
   <Resource name="jdbc/juddiDB" auth="Container"
             type="javax.sql.DataSource"
             initialSize="30" maxActive="100"
             maxIdle="30" maxWait="10000"
             username="root" password="adria"
             driverClassName="com.mysql.jdbc.Driver"
             removeAbandoned="true"
             removeAbandonedTimeout="60"
             logAbandoned="true"
             url="jdbc:mysql://localhost:3306/juddi" />
</Context>
```

Astfel, am stabilit o serie de parametri referitori la conexiunea cu serverul de baze de date via *driver*-ul mai sus menționat.

În cazul în care lucrați cu Tomcat versiunea 4.1.24, acest fișier are forma:

```
<Context path="/juddi" docBase="juddi" debug="5"
         reloadable="true" crossContext="true">
   <Logger
      className="org.apache.catalina.logger.FileLogger"
      prefix="localhost_juddiDB_log" suffix=".txt"
      timestamp="true" />
   <Resource name="jdbc/juddiDB"
             auth="Container"
             type="javax.sql.DataSource" />
          <ResourceParams name="jdbc/juddiDB">
            <parameter>
              <name>factory</name>
              <value>
   org.apache.commons.dbcp.BasicDataSourceFactory
              </value>
            </parameter>
            <parameter>
              <name>maxActive</name>
              <value>100</value>
```

```
            </parameter>
            <parameter>
              <name>maxIdle</name>
              <value>30</value>
            </parameter>
            <parameter>
              <name>maxWait</name>
              <value>10000</value>
            </parameter>
            <parameter>
              <name>username</name>
              <value>root</value>
            </parameter>
            <parameter>
              <name>password</name>
              <value>adria</value>
            </parameter>
            <parameter>
               <name>driverClassName</name>
               <value>org.gjt.mm.mysql.Driver</value>
            </parameter>
            <parameter>
              <name>url</name>
              <value>
   jdbc:mysql://localhost:3306/jUDDI?autoReconnect=true
              </value>
            </parameter>
   </ResourceParams>
</Context>
```

Sunteți în acest moment pregătiți să rulați *jUDDI*.

Pentru a verifica modul de execuție a instrumentului *jUDDI*, porniți serverul Web și introduceți adresa *http://localhost:8080/juddi/happyjuddi.jsp*. Dacă obțineți informații similare celor din figura de mai jos, atunci din acest moment registrul dumneavoastră UDDI este pregătit să recepționeze cereri.

Pachetul de instalare *jUDDI* include o serie de alte exemple utile pe care le puteți folosi pentru a trimite cereri registrului.

Folosind succesiunea de pași detaliată anterior, puteți utiliza *jUDDI* pentru crearea altor regiștri privați UDDI, care să poată deservi cereri de publicare, autentificare și interogare.

Nu ne rămâne decât să vă urăm succes în exploatarea registrului UDDI propriu!

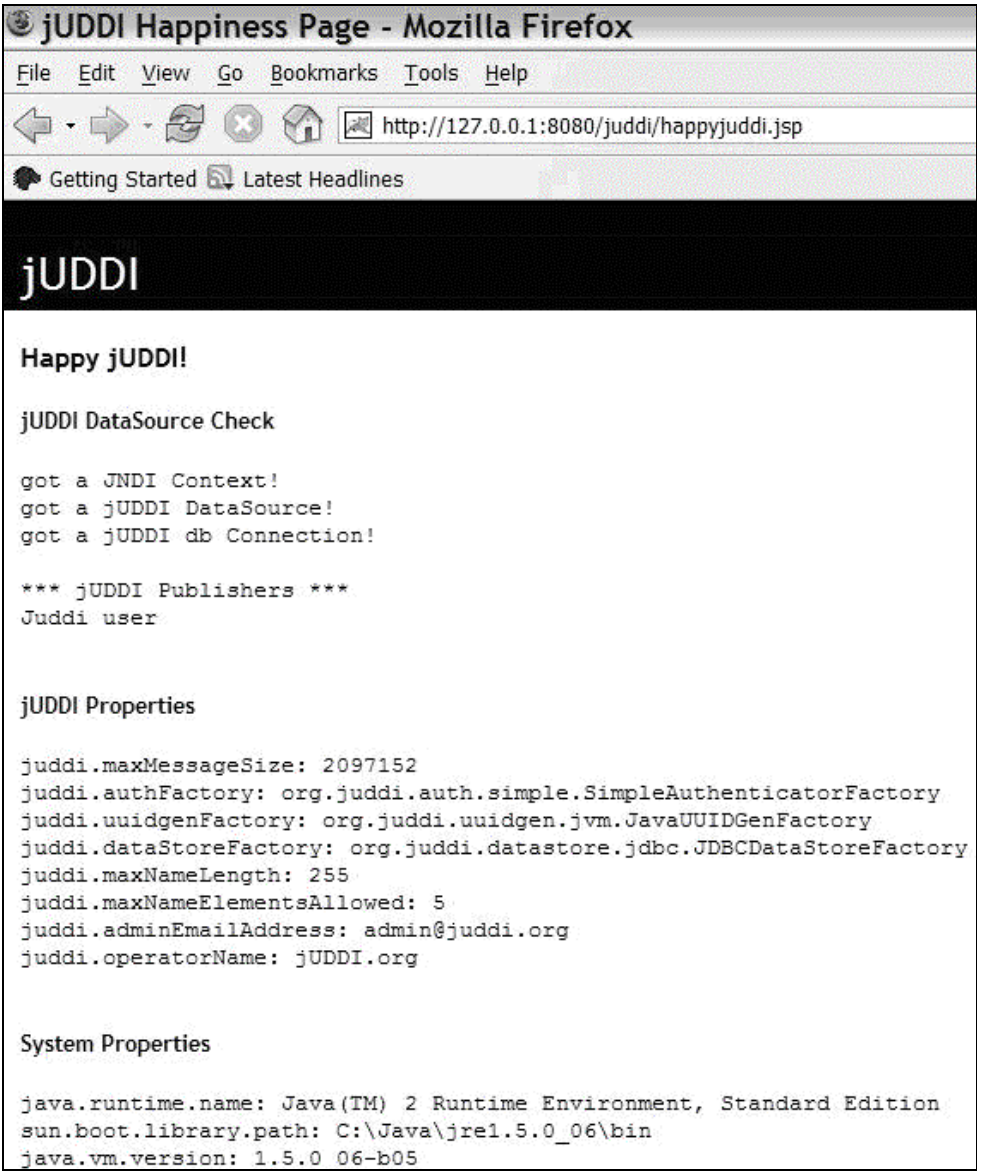


Figura 10. *Mesajele de stare obținute în urma rulării aplicației de diagnosticare* jUDDI

2. WSIL (*Web Service Inspection Language*)

Inițiativa WSIL (*Web Service Inspection Language*) – regăsită și sub numele *WS-Inspection* – definește modul în care se poate descoperi o descriere XML a unui serviciu Web aflat pe un server Web. Astfel, serviciile Web pot fi regăsite

facil. Via WSIL, un furnizor de servicii poate să-și facă disponibile serviciile pentru a putea fi descoperite și utilizate de către consumatori.

Acest aspect este similar cu misiunea UDDI, dar WSIL se dorește a fi o etapă superioară UDDI în evoluția procesului de regăsire a serviciilor Web.

Figura următoare ilustrează noile relații ce pot fi stabilite între consumatori, furnizori și regiștrii centralizați UDDI. Prima diferență se materializează în faptul că o cerere de căutare a unui serviciu Web nu se realizează direct asupra regiștrilor centralizați UDDI. În schimb, căutarea este efectuată de către consumatori direct la furnizorul de servicii.

Conform acestui model, regiștrii UDDI centralizați și nemoderați vor deveni descentralizați și moderați. De observat, de asemenea, că nimeni nu împiedică furnizorii de servicii să-și înregistreze serviciile în cadrul regiștrilor UDDI și să redirecteze găsirea serviciului spre regiștrii UDDI.

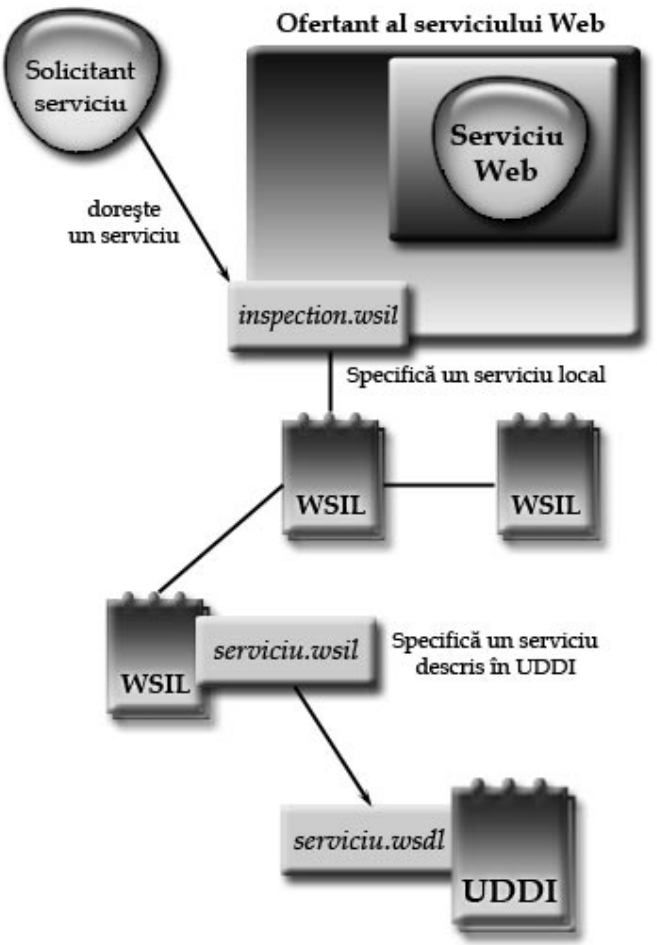


Figura 11. *Furnizorii de servicii își fac serviciile cunoscute via WSIL*

Trebuie să observăm că WSIL nu reprezintă un limbaj de descriere a serviciilor Web, această responsabilitate având-o în continuare WSDL. Limbajul WSIL este folosit doar pentru a face publice serviciile Web.

Elementele limbajului WSIL aparţin spaţiului de nume desemnat de adresa *http://schemas.xmlsoap.org/ws/2001/10/inspection/*.

Documentul WSIL minimal constă dintr-un element-rădăcină *service*, care conţine un element *description*. Elementul *service* are rol de prezentare a serviciilor. Documentele WSIL nu sunt folosite doar pentru descrierea serviciilor Web via WSDL, astfel încât să poată fi utilizate şi alte maniere de definire a serviciilor.

Specificaţiile WSIL definesc o serie de convenţii care să permită consumatorilor să localizeze documentele WSIL pe orice sit Web (vezi şi figura 11). Aceste convenţii presupun existenţa unui nume prestabilit al documentului WSIL şi o manieră de legătură la un document WSIL. Numele prestabilit este *inspection.wsil*. Un document cu un astfel de nume ar trebui plasat în punctele de intrare cele mai vizitate ale unui sit Web. De exemplu, dacă un astfel de punct este *http://www.magazin-portocale.info/*, atunci documentul WSIL ar trebui localizat prin *http://www.magazin-portocale.info/inspection.wsdl*.

Însă foarte puţini furnizori ce recurg la WSIL vor oferi documentul WSIL la acest nivel. Putem să ne gândim că aceste documente WSIL pot fi regăsite destul de uşor cu ajutorul unor motoare de căutare precum Google sau Yahoo!.

Având scopul de a permite furnizorilor să-şi organizeze serviciile într-o manieră ierarhică, documentele WSIL pot fi interconectate. Aceasta se realizează prin prezenţa elementului *link*. Un document WSIL poate avea un număr oricât de mare de legături, generându-se astfel o întreagă reţea de documente WSIL.

Drept exemplificare, vom oferi un fragment din documentul WSIL pus la dispoziţie de situl *XMethods*, document localizat la adresa *http://www.xmethods.net/inspection.wsil*. Arborele asociat din figura 12 a fost generat cu ajutorul instrumentului *oXygen XML Editor*.

În ciuda prezenţei regiştrilor UDDI, se remarcă existenţa unui „gol" (*gap*) interpus între furnizorii de servicii şi consumatori. Specificaţia WSIL propusă de IBM şi Microsoft încearcă să acopere acest gol, în timp ce UDDI se perfecţionează şi va deveni o soluţie convenabilă mai ales în contextul *enterprise*.

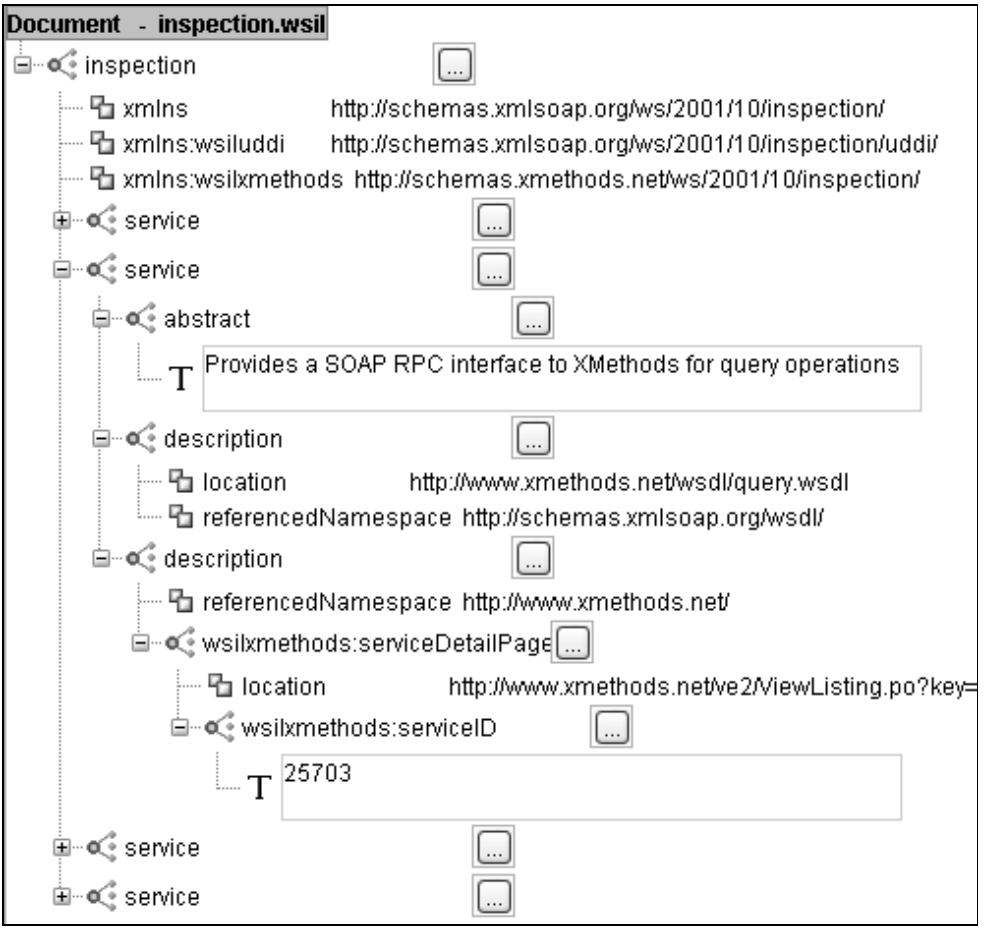


Figura 12. *Detalii privind un serviciu Web pus la dispoziție de XMethods și descris prin WSIL*

Referințe

Booth, D. *et al.* (eds.), *Web Services Architecture*, W3C Working Draft, Boston, 2003: http://www.w3.org/TR/ws-arch/

Cabrera, L.F., Kurt, C., *Web Services Architecture and Its Specifications*, Microsoft Press, 2005

Clement, L. *et al.* (eds.), *UDDI Version 3.0*, OASIS Open, 2004: http://uddi.org/pubs/uddi_v3.htm

Guruge, A., *Web Services: Theory and Practice*, Digital Press, 2004

Tanasă, Ș., Olaru, C., *Dezvoltarea aplicațiilor Web folosind Java*, Polirom, Iași, 2005

Weerawarana, S. *et al.*, *Web Services Platform Architecture*, Prentice Hall PTR, 2005

* * *, *IBM's UDDI Business Registry*: http://www.uddi.ibm.com/
* * *, *Instrumentul jUDDI*: http://ws.apache.org/juddi/
* * *, *Java Coffee-Break*: http://www.javacoffeebreak.com/
* * *, *Microsoft's UDDI Business Registry*: http://uddi.microsoft.com/
* * *, *Serverul Apache*: http://httpd.apache.org/
* * *, *Situl dedicat dezvoltatorilor MySQL*: http://dev.mysql.com/
* * *, *Situl dedicat serverului Tomcat*: http://jakarta.apache.org/tomcat/
* * *, *Situl dedicat serviciilor Web*: http://www.webservices.org/
* * *, *Situl Sun privitor la Java*: http://java.sun.com/
* * *, *Situl XML SOAP*: http://www.xmlsoap.org/
* * *, *Source Forge*: http://www.sourceforge.net/
* * *, *UDDI (Universal Description, Discovery, and Integration):* http://www.uddi.org/
* * *, *WS-I (Web Services-Interoperability Organization)*: http://www.ws-i.org/
* * *, *XMethods*: http://www.xmethods.net/

Capitolul 6

Dezvoltarea și utilizarea serviciilor Web

„Dacă teoria mea nu corespunde faptelor, cu atât mai rău pentru fapte."
(Georg Wilhelm Friedrich Hegel)

1. Dezvoltarea și invocarea de servicii Web folosind instrumente *open source*

1.1. *Introducere*

În majoritatea cazurilor, serviciile Web funcționează conform paradigmei client/server: serverul rulează permanent, unul sau mai mulți clienți trimit spre acesta cereri SOAP printr-un protocol de transport (cel mai uzual, HTTP), iar serverul întoarce un răspuns.

În cazul serviciilor Web, standardele sunt esențiale pentru asigurarea inter-operabilității. Așa cum s-a văzut în capitolele anterioare, codificarea conținutului mesajelor vehiculate (cererea și răspunsul) prin SOAP adoptă formatul XML, iar descrierea serviciului Web se bazează pe limbajul WSDL. Eventual, pot fi atașate opțional date codificate în stilul *base64*, folosind propunerea SwA (*SOAP with Attachments*), ori metoda standardizată, recomandată de Consorțiul Web – MTOM (*Message Transmission Optimization Mechanism*). Mecanismul de atașare respectă bine-cunoscutele prevederi MIME (*Multipurpose Internet Mail Extensions*), larg folosite în cazul mesajelor de poștă electronică.

Trebuie să remarcăm încă o dată că nu există nici o restricție în alegerea limbajului de programare în care se vor implementa serviciul Web și posibilii lui clienți (programe *desktop*, aplicații Web, alte servicii Web etc.). Indiferent ce limbaj, instrument, platformă sunt folosite, în esență dezvoltarea și utilizarea serviciilor Web prezintă aceleași caracteristici și adoptă șabloane de proiectare similare.

Unele instrumente SOAP implementează propriul server Web, folosind HTTP (de exemplu, *SOAP::Lite* sau *Microsoft Visual Web Developer*). Altele, precum

extensia *Soap* pentru PHP sau biblioteca *gSOAP*, presupun instalarea în cadrul unui server Web particular, astfel încât *daemon*-ul HTTP va manevra mesajele SOAP spre o componentă *proxy*, care realizează invocarea serviciului Web în manieră transparentă pentru programator. O serie de instrumente SOAP suportă un mecanism de transport configurabil care pemite selecția, la nivel de aplicație, a protocolului de transport. Modulul *SOAP::Lite*, de pildă, oferă suport pentru HTTP, FTP, SMTP sau Jabber.

Instrumentele SOAP furnizează o componentă *proxy*, menită a procesa și interpreta mesajul SOAP în vederea invocării codului aplicației. Această componentă trebuie să aibă în responsabilitate recunoașterea stilului de codificare, translatarea datelor din format nativ – de exemplu obiecte Java sau .NET – în XML (și invers), procesarea antetului din mesajul SOAP cu atributul `mustUnderstand="true"` etc. Atunci când o componentă *proxy* manevrează un mesaj SOAP, va trebui să efectueze următoarele:

- deserializarea mesajului, dacă este necesar, din format XML într-o formă adecvată pentru a-l putea analiza;
- invocarea codului (a metodelor);
- serializarea răspunsului primit în XML și manevrarea lui în scopul trimiterii acestuia la emițător.

În ciuda diferențelor de implementare, instrumentele pentru dezvoltarea de servicii Web prin SOAP urmează același model.

În cadrul acestui subcapitol ne propunem să detaliem modul de implementare și de invocare a unor servicii Web pe baza uneltelor *open source* existente pentru PHP și Perl. Pentru început, vom construi un server și un client SOAP simplu în PHP, utilizând extensia *Soap* disponibilă în versiunea 5 a acestui server de aplicații și, ca alternativă, biblioteca *NuSOAP* care poate fi folosită și în PHP 4. În partea a doua a acestui subcapitol ne propunem să scriem un serviciu Web oferind o interfață comodă pentru operațiile uzuale realizate asupra resurselor unui sistem de fișiere aflat de distanță. Pentru aceasta, vom recurge la modulul *SOAP::Lite*, unul dintre cele mai cunoscute instrumente de dezvoltare în limbajul Perl a serviciilor Web și a clienților aferenți acestora.

1.2. *Furnizarea stocurilor de portocale via un serviciu Web scris în PHP 5*

Misiunea pe care o avem de realizat este ușor de îndeplinit. Trebuie să implementăm un server SOAP care va oferi accesul la un serviciu Web, incluzând o singură metodă – *furnizeazaCantit()*. Aceasta va avea ca parametru de intrare un

şir de caractere desemnând un sortiment de portocale şi va returna cantitatea existentă, dacă acel sortiment există. Nu vom furniza o descriere WSDL asociată serviciului, invocarea putându-se realiza explicit, după cum vom vedea mai jos. Codul-sursă al acestui serviciu simplu este dat în continuare:

```
<?php
try {
    // nu oferim nici o descriere WSDL,
    // stabilim URI-ul serviciului
    $server = new SoapServer(null,
       array('uri' => 'urn:portocale.info'));
    // adăugăm metodele implementate
    $server->addFunction('furnizeazaCantit');
    // aşteptăm cereri SOAP
    $server->handle();
}  catch (SOAPFault $exception) {
    echo 'A apărut o excepţie: ' . $exception;
}

// funcţie care furnizează cantitatea
// dintr-un sortiment de portocale
function furnizeazaCantit ($numeSortiment) {
  // de obicei, aici vom efectua o interogare SQL
  // ori una XQuery
  switch ($numeSortiment) {
    case 'japoneze': return 33;
    case 'albastre': return 74;
    default        : return 'inexistent';
  }
}
?>
```

Primul argument al constructorului desemnează adresa (URL-ul) documentului WSDL ce descrie serviciul, în acest caz fiind *null*. Spaţiul de nume prin care e identificat serviciul dezvoltat este dat în situaţia de faţă de un URN – *urn:portocale.info*. Fiecare funcţie implementată va fi adăugată cu *addFunction()*. De asemenea, unui serviciu i se poate asocia o clasă prin intermediul metodei *setClass()*. Cererile SOAP vor fi rezolvate via *handle()*, care va avea în responsabilitate şi (de)serializarea datelor.

Deoarece extensia *Soap* este inclusă în distribuţia standard PHP 5, pentru rularea programului de mai sus nu va trebui instalat nimic suplimentar. În exemplificările de faţă am recurs la PHP versiunea 5.0.4 sub diverse versiuni ale serverului Apache 2.0.

1.3. *Un client PHP pentru serviciul Web privitor la portocale*

Recurgând tot la extensia *Soap*, vom concepe un client care va invoca metoda expusă de serviciul Web pentru o serie de sortimente de portocale. Liniile de cod PHP sunt următoarele:

```
try {
   $client = new SoapClient(null,
      array('location' => 'http://localhost/sxml/
                           oranges-server.php',
            'uri'      => 'urn:portocale.info'));
   // realizăm o suită de invocări ale metodei dorite
   foreach (array('albastre', 'japoneze', 'celeste')
     as $sortiment) {
     $rez = $client->__call('furnizeazaCantit',
                            array($sortiment));
     echo "<p>Stocul de portocale $sortiment
       este <strong>$rez</strong>.</p>";
   }
} catch (SOAPFault $exception) {
   echo 'A apărut o excepție: ' .
     $exception->faultstring;
}
```

Deoarece nu avem la dispoziție descrierea WSDL a serviciului, vom folosi *__call()* pentru invocarea metodei *furnizeazaCantit()*. Excepțiile ce pot surveni sunt capturate grație construcției *try … catch* incluse în PHP 5. Obiectul *SOAPFault* încapsulează informațiile privitoare la un *fault* SOAP; în cazul de față am utilizat proprietatea *faultstring* pentru a afișa mesajul de excepție. Eșuarea unui apel SOAP poate fi verificată și cu *is_soap_fault()*. Mesajele vehiculate vor respecta în mod implicit versiunea 1.1 a protocolului SOAP. Extensia *Soap* oferă posibilitatea de a recurge însă și la SOAP 1.2.

În urma iterațiilor realizate, vom obține un rezultat precum cel din figura 1. Tipul răspunsului primit este unul eterogen (fie întreg, fie *string*), în funcție de existența sortimentului de portocale pentru care realizăm invocarea metodei.


Portocale - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://endirra.ro/oranges-client.php
Go
Disable Cookies CSS Forms Images Information Miscellaneous
Stocul de portocale albastre este 74.
Stocul de portocale japoneze este 33.
Stocul de portocale celeste este inexistent.
Done
0 errors / 0 warnings

Figura 1. *Invocarea serviciului Web implementat în PHP 5*

1.4. *Un serviciu de manipulare a fişierelor implementat în Perl cu* SOAP::Lite

Ne propunem în continuare să implementăm un server SOAP, disponibil la portul 3333 via protocolul HTTP, care va încapsula funcţionalităţile unui serviciu Web, oferind diverse operaţii asupra fişierelor existente pe server. Drept exemplificare, vom scrie următoarele metode:

– *List*, care furnizează lista fişierelor existente la nivel de server; nu necesită parametri de intrare şi întoarce o structură de forma `<file name="..." />` pentru fiecare fişier găsit (vezi *infra*);
– *CheckIfExists*, ce verifică existenţa unui fişier la nivel de server, având ca parametru de intrare un şir de caractere desemnând un nume de fişier; va întoarce valoarea 1 dacă fişierul există şi 0 dacă e inaccesibil (nu poate fi regăsit ori nu există suficiente drepturi de acces asupra lui);
– *Rename*, care redenumeşte un fişier, necesitând doi parametri (numele vechi şi cel nou) de tip şir de caractere; va întoarce un şir de caractere: „*success*” în caz de succes şi „*error*” în caz contrar.

Codul-sursă al programului Perl este următorul:

```
#!/usr/bin/perl -w
use strict;
use SOAP::Transport::HTTP +debug => [qw(all)];

# nu dorim să fim întrerupţi de semnale
# precum 'Broken pipe' ori la acţionarea
```

```
# combinației de taste CTRL+C
$SIG{PIPE} = $SIG{INT} = 'IGNORE';

# inițializăm serverul, folosind un daemon HTTP
my $server = SOAP::Transport::HTTP::Daemon->new (
   LocalAddr => 'localhost',    # adresa locală
   LocalPort => 3333,           # portul utilizat
   Reuse     => 1);             # reutilizăm portul
# funcționalitatea serverului este furnizată
# de clasa 'XMLFiles'
$server->dispatch_to ('XMLFiles');

print 'Serverul SOAP rulează la ' . $server->url;
# ne blocăm, așteptând cereri din partea clienților
$server->handle;

# clasa implementând funcționalit. serviciului Web
BEGIN {
package XMLFiles;
use SOAP::Lite;
use base qw(SOAP::Server::Parameters);

sub List {  # metoda de generare a listei
            # fișierelor existente pe server
    my $self = shift;
    my $envelope = pop;
    my $files = '';
    # returnăm răspunsul drept frag. de doc. XML
    foreach my $file (glob('*')) {
       $files .= "<file name=\"$file\" />";
    }
    return SOAP::Data->type('xml' => $files);
}

sub CheckIfExists { # metoda de verificare
                    # a existenței unui fișier
    my $self = shift;
    my $envelope = pop;
    # preluăm param. de intrare via o expresie XPath
    # (nu știm cum ar putea fi încapsulat)
    my $filename = $envelope->dataof('//filename');
    # trimitem un 'fault'
    # dacă nu există parametrul de intrare
    die SOAP::Fault->faultcode('Server.CheckIfExists')
                   ->faultstring(
                         'Input parameter is missing')
      unless $filename;
```

```
    # furnizăm fişierul în cauză, dacă există
    return (-e $filename->value) ? 1 : 0;
}

sub Rename { # metoda de redenumire a unui fişier
    my $self = shift;
    my $envelope = pop;
    # preluăm parametrii de intrare
    my $filename = $envelope->dataof('//filename');
    my $newname  = $envelope->dataof('//newname');
    # eventual, semnalăm inexistenţa lor
    die SOAP::Fault->faultcode('Server.Rename')
                   ->faultstring(
              'Input filename parameter is missing')
      unless $filename;
    die SOAP::Fault->faultcode('Server.Rename')
                   ->faultstring(
              'Input newname parameter is missing')
      unless $newname;
    # returnăm răspunsul
    return SOAP::Data->name('result' =>
      (rename ($filename->value, $newname->value) ?
         'success' : 'error'));
}

# eventual, alte metode...
}
```

Serverul va rula la infinit, ocupând portul 3333 al maşinii locale (vezi figura 2). Va pune la dispoziţie via SOAP 1.1 peste HTTP cele trei metode descrise mai sus. Pentru fiecare metodă în parte, preluăm parametrii necesari. În cazul în care aceştia nu sunt furnizaţi de client, se va întoarce un mesaj de tip *fault* SOAP, semnalându-se o situaţie de excepţie. Am dorit ca răspunsul oferit să fie de mai multe tipuri de date (întreg, şir de caractere sau o listă de fişiere marcată în XML), pentru a se putea observa diferenţele de codificare.

Modulul *SOAP::Perl* apare în majoritatea distribuţiilor Perl actuale. Versiunile utilizate pentru testarea şi rularea exemplelor din această carte au fost 0.65 şi 0.68. În cazul în care modulul este indisponibil, poate fi preluat din CPAN (*Comprehensive Perl Archive Network*) – accesibil la *http://search.cpan.org/* – şi instalat conform următorilor paşi:

```
(infoiasi)$ perl Makefile.PL
(infoiasi)$ make
(infoiasi)$ make test
(infoiasi)$ make install
```

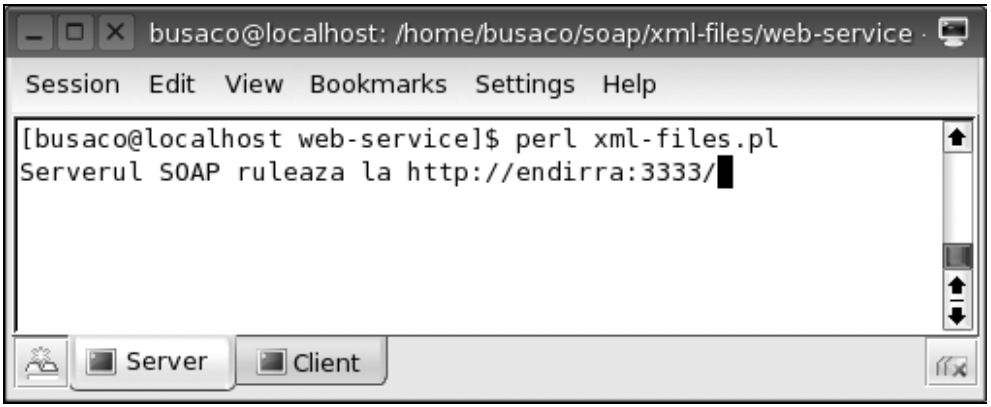


Figura 2. *Rularea serverului SOAP într-o fereastră a unei console KDE*

Pentru a putea realiza transferuri SOAP cu fişiere ataşate (*SOAP with Attachments*) va trebui să recurgem la modulul *SOAP::MIME*, care îmbunătăţeşte o serie de aspecte ale *SOAP::Lite*. Mai trebuie remarcat faptul că modulul *SOAP::Lite* nu oferă suport pentru generarea automată de descrieri WSDL corespunzătoare serviciilor Web implementate.

Pentru alte detalii privitoare la folosirea sesiunilor în cadrul serviciilor Web, a invocării unor servicii externe şi a utilizării unor metode de autentificare în cadrul serviciilor Web dezvoltate, cititorii interesaţi pot consulta documentaţiile şi exemplele incluse în arhiva de instalare a modulului *SOAP::Lite*.

1.5. *Scrierea în Perl a clientului SOAP*

În acest moment putem scrie un client Perl simplu care să invoce metodele serviciului Web implementat. Vom realiza o interacţiune de bază cu utilizatorul, la nivelul liniei de comandă (*prompt*-ul interpretorului de comenzi, de obicei *bash* în Linux). Codul-sursă al programului-client e furnizat mai jos:

```
#!/usr/bin/perl -w
# folosim modulul SOAP::Lite cu opţiunea de depanare
use SOAP::Lite +debug => [qw(all)];

if (scalar(@ARGV) < 1) {
   print "Client de accesare a metodelor oferite de un
serviciu Web\npentru accesarea sistemului de fişiere de pe
server.\n";
   print "Sintaxa: $0 <fişier_existent_pe_server>\n";
   exit(1);
}
```

```
# ne conectăm la server
my $soap = SOAP::Lite
        ->uri('http://127.0.0.1:3333/XMLFiles/')
                  # adresa spațiului de nume
        ->proxy(
        'http://127.0.0.1:3333/XMLFiles/xml-files.pl');
                  # locația serviciului

# împachetăm explicit parametrii,
# folosind nume calificate
# pentru parametrii metodei invocate
my @params = (SOAP::Data->name('filename', $ARGV[0]));

# verificăm dacă fişierul există pe server
$response = $soap->call('CheckIfExists' => @params);

# verificăm dacă a apărut o eroare (SOAP fault)
if ($response->fault) {
   die 'A apărut o eroare: ' .
         $response->faultstring;
}
# dacă rezultatul primit e nul, fişierul nu există
if (!$response->result) {
   die 'Fişier inaccesibil (nu există pe server?)';
}

# invocăm metoda de redenumire a fişierului
# adăugând şi numele cel nou al acestuia
push (@params, SOAP::Data->name('newname',
                $ARGV[0] . '.renamed'));

# afişăm rezultatul execuției metodei
print 'Redenumire realizată cu ' .
  $soap->call('Rename' => @params)->result;
```

Programul va verifica existența unui fișier al cărui nume este furnizat de utilizator în linia de comandă, va încerca să-l redenumească (numele vechi va fi sufixat cu *.renamed*), apoi va afișa lista fișierelor existente pe server (putem verifica astfel dacă redenumirea s-a efectuat).

În urma execuției acestui *script*, vom putea obține un rezultat asemănător cu următorul (presupunem că pe server există fișierul denumit *portocale.txt*):

```
(infoiasi)$ perl xml-files-client.pl portocale.txt
Fisierul exista.
Redenumire realizata cu succes.
Fisierele existente pe server:
```

```
portocale.txt.renamed
xml-files.pl
xml-files.pl~
```

Prima cerere efectuată de programul-client este încapsulată într-un mesaj HTTP de forma de mai jos, în care se pot observa câmpurile-antet HTTP, tipul conținutului (fișier XML) și corpul mesajului SOAP (reprezintă invocarea metodei *CheckIfExists* cu parametrul corespunzător furnizat tot sub forma de marcaje XML):

```
POST http://127.0.0.1:3333/XMLFiles/xml-files.pl HTTP/1.1
Accept: text/xml
Accept: multipart/*
Accept: application/soap
Content-Length: 517
Content-Type: text/xml; charset=utf-8
SOAPAction:
  "http://127.0.0.1:3333/XMLFiles/#CheckIfExists"

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc=
     "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   soap:encodingStyle=
     "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soap=
     "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <namesp1:CheckIfExists
    xmlns:namesp1="http://127.0.0.1:3333/XMLFiles/">
    <filename xsi:type="xsd:string">
        portocale.txt
    </filename>
  </namesp1:CheckIfExists>
</soap:Body>
</soap:Envelope>
```

Răspunsul oferit de server este:

```
HTTP/1.1 200 OK
Date: Fri, 04 Aug 2006 18:25:51 GMT
Server: libwww-perl-daemon/1.36
Content-Length: 523
Content-Type: text/xml; charset=utf-8
Client-Date: Fri, 04 Aug 2006 18:25:51 GMT
```

```
Client-Peer: 127.0.0.1:3333
Client-Response-Num: 1
SOAPServer: SOAP::Lite/Perl/0.65_5

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc=
     "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   soap:encodingStyle=
     "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soap=
     "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
   <namesp9:CheckIfExistsResponse
      xmlns:namesp9="http://127.0.0.1:3333/XMLFiles/">
      <s-gensym27 xsi:type="xsd:int">1</s-gensym27>
   </namesp9:CheckIfExistsResponse>
</soap:Body>
</soap:Envelope>
```

Se poate remarca faptul că valoarea răspunsului furnizat (întregul 1) este încapsulată în marcatorii XML din cadrul corpului plicului SOAP ce împachetează informațiile vehiculate între serviciu și clienții săi.

Lucrurile decurg similar pentru a doua cerere – corespunzătoare invocării metodei *Rename* – și răspunsul aferent.

Pentru a treia cerere, de invocare a metodei *List*, nu există argumente de intrare, în cadrul corpului plicului SOAP apărând doar:

```
<namesp3:List xsi:nil="true"
   xmlns:namesp3="http://127.0.0.1:3333/XMLFiles/" />
```

Rezultatul metodei reprezintă o structură XML de genul:

```
<namesp4:ListResponse
   xmlns:namesp4="http://127.0.0.1:3333/XMLFiles/">
   <file name="portocale.txt.renamed" />
   <file name="xml-files.pl" />
</namesp4:ListResponse>
```

Spațiile de nume de forma *namespN* sunt generate în mod automat de către *SOAP::Lite* pentru a indica apartenența fiecărui element XML din cadrul mesajului SOAP vehiculat.

În cazul în care clientul nu furnizează numele parametrului corect (de exemplu, în loc de *filename* se trimite *file* pentru metoda *CheckIfExists*), se va obține un mesaj de tip *fault* SOAP emis de serviciu (codul entității emitente, șirul explicativ și adresa entității care a cauzat excepția):

```
...
<soap:Body>
   <soap:Fault>
      <faultcode>
          soap:Server.CheckIfExists
      </faultcode>
      <faultstring>
          Input parameter is missing
      </faultstring>
      <faultactor>
          http://127.0.0.1:3333/
      </faultactor>
   </soap:Fault>
</soap:Body>
...
```

1.6. *Invocarea serviciului de către un client PHP*

Am dori ca serviciul să poată fi invocat de o aplicație Web, astfel încât interacțiunea cu utilizatorul să poată fi realizată prin intermediul unui *browser* Web. Aceasta o vom realiza scriind un client PHP, inter-operabilitatea având loc fără probleme, din moment ce mesajele sunt vehiculate via XML, indiferent de platformă. Programul PHP recurge la biblioteca *NuSOAP* (versiunea 0.6.5) și are următorul cod-sursă:

```
<?php
require_once('lib/nusoap.php'); # utilizăm NuSOAP

// instanțierea unui client SOAP
// pe baza descrierii serviciului Web
$client = new soapclient(
            'http://endirra.ro:3333/xml-files.pl');
// preluăm posibilele erori
$err = $client->getError();
if ($err) {
   // semnalăm eroarea
   echo '<h3>Eroare de inițializare</h3><pre>'
         . $err . '</pre>';
}
// stabilim parametrii de intrare ai metodei invocate
```

```
$param = array('filename' => 'xml-files.pl');
                # există cu siguranţă
# de încercat cu un fişier inexistent:
# $param = array('filename' => 'inexistent.xml');
// stabilim spaţiul de nume folosit
$namespace = 'http://127.0.0.1:3333/XMLFiles/';

// realizăm apelul
$result = $client->call('CheckIfExists',
  array('parameters' => $param), $namespace, '', true);
// verificăm dacă există vreun fault
if ($client->fault) {
   echo '<h3>Fault</h3><pre>';
   print_r($result);
   echo '</pre>';
} else { // verificăm dacă sunt erori
   $err = $client->getError();
   if ($err) {
     echo '<h3>Eroare</h3><pre>' . $err . '</pre>';
   } else {
     // afişăm rezultatul
     echo '<h3>Rezultat</h3>';
     echo '<p>Fişierul ' . ($result ?
               'există' : 'e inaccesibil') . '.</p>';
   }
}

// cererea, răspunsul şi informaţiile de depanare
echo '<h3>Cerere</h3><pre>' .
  htmlspecialchars($client->request, ENT_QUOTES) .
  '</pre>';
echo '<h3>Răspuns</h3><pre>' .
  htmlspecialchars($client->response, ENT_QUOTES) .
  '</pre>';
echo '<h3>Depanare</h3><pre>' .
  htmlspecialchars($client->debug_str, ENT_QUOTES) .
  '</pre>';
?>
```

După cum se poate remarca, se instanţiază un obiect *$client* al clasei *soapclient()* pusă la dispoziţie de biblioteca mai sus menţionată. Constructorul necesită adresa Web a serviciului dorit a fi invocat. Posibilele erori survenite pot fi captate cu metoda *getError()*, iar *fault*-urile SOAP via proprietatea *fault*.

În acest caz am invocat doar metoda de verificare a existenţei unui fişier, recurgând la metoda *call()*. Convenţia este ca parametrii transmişi să fie încapsulaţi într-un tablou (*array*).

Un posibil rezultat este ilustrat de figura 3, în care se remarcă și afișarea cererii, a răspunsului și a informațiilor privitoare la depanare, direct în programul de navigare Web. Programul-client PHP și serverul SOAP scris în Perl au fost testate pe platformele Linux și Windows. Navigatorul Mozilla Firefox care realizează invocarea *script*-ului PHP rulează în Windows.

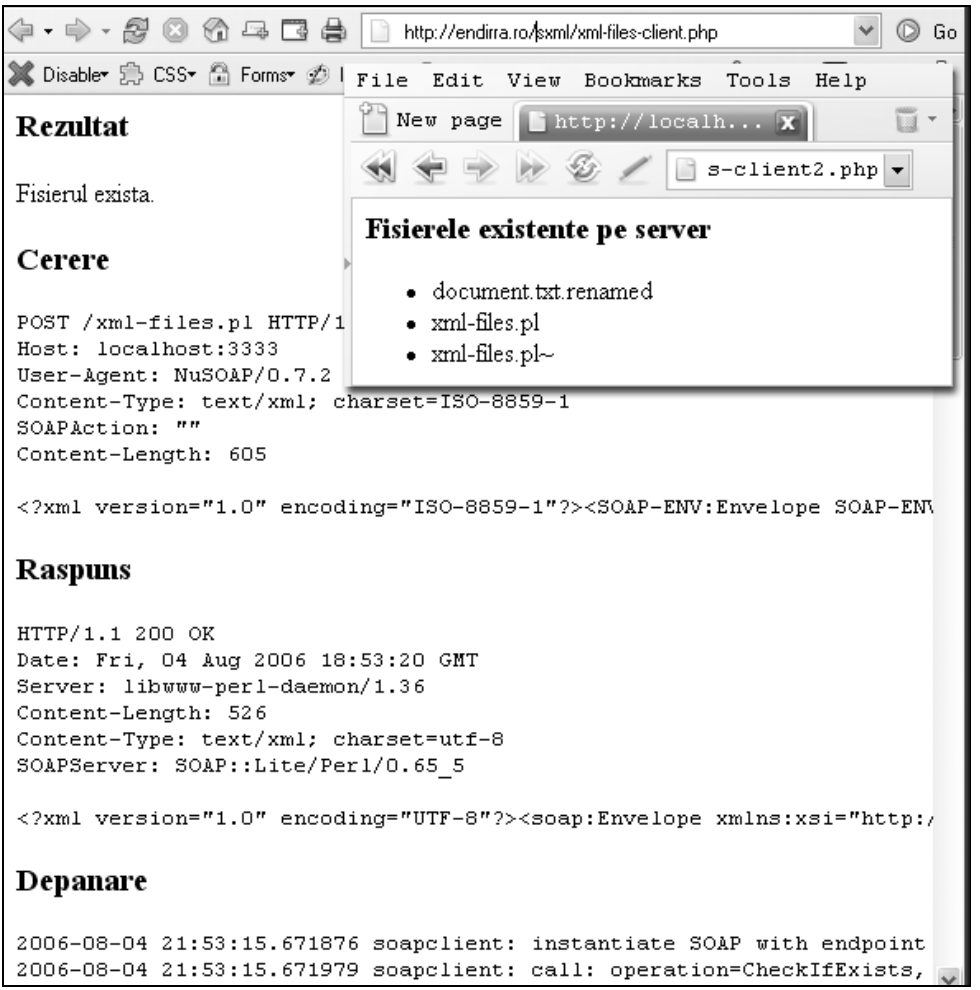


Figura 3. *Rezultatele execuției programelor PHP ce invocă metodele* CheckIfExists *și* List *ale serviciului Web implementat în Perl (au fost folosite navigatoarele Mozilla Firefox și Opera)*

În cazul invocării metodei *List*, care întoarce drept rezultat un fragment de document XML, vom procesa datele obținute prin intermediul extensiei *Simple*

XML (disponibilă în PHP versiunea 5). Liniile de cod prezentând interes în acest context sunt următoarele:

```
// realizăm apelul
$result = $client->call('List', undefined,
  $namespace, '', true);
$err = $client->getError();
if ($err) {
  echo '<h3>Eroare</h3><pre>' . $err . '</pre>';
} else {
  // afişăm lista de fişiere
  echo '<h3>Fişierele existente pe server</h3><ul>';
  // procesăm răspunsul SOAP via Simple XML
  $xml = simplexml_load_string ($client->responseData);
  foreach ($xml->xpath("//file") as $file) {
    // afişăm fiecare nume de fişier
    echo '<li>' . $file['name'] . '</li>';
  }
  echo '</ul>';
}
```

Rezultatul obținut este ilustrat tot de figura 3.

2. Inter-operabilitatea între diverse tehnologii, instrumente și limbaje de programare a serviciilor Web

2.1. *Dezvoltarea de servicii Web în* .NET Framework

În cadrul platformei .NET, un serviciu Web este considerat ca fiind un tip specific de *assembly* (formă atomică a unui modul de cod), tratat într-o manieră specială de către mediu. Serviciile Web sunt conținute de fișiere cu extensia *.asmx*. Serverul Web Microsoft IIS (*Internet Information Services*) recunoaște fișierele *.asmx* ca fiind servicii Web și în mod automat exportă funcțiile *assembly*-ului referențiat.

Procesul de dezvoltare și exploatare a unui serviciu Web în .NET este (relativ) simplu: se scrie codul-sursă, se salvează într-un fișier *.asmx*, se exportă respectivul fișier într-un director virtual al serverului IIS și se invocă respectivul serviciu via un client.

Un serviciu Web simplu implementat în C#

Să începem cu un serviciu Web extrem de simplu. Conform tradiției, primul program scris trebuie să fie unul care afișează „Salut, lume!" („*Hello, world!*"). Vom scrie un serviciu care să furnizeze mesajul „Bună *X*", unde *X* reprezintă un șir de caractere introdus de utilizator (printr-un anumit client, desigur). Putem recurge la orice editor de texte favorit pentru a crea fișierul de cod C# cu numele *salutari.asmx*.

Liniile programului nostru sunt următoarele:

```
<%@ WebService Language="C#" Class="Hello" %>
using System.Web.Services;

[WebService(Namespace="urn:Hello")]
public class Hello {
  [WebMethod]
  public string sayHello (string nume)   {
    return "Buna " + nume;
  }
}
```

Linia *<%@ WebService Language="C#" Class="Hello" %>* este o directivă care indică mediului .NET să exporte un serviciu Web, scris în limbajul C# și implementat de clasa cu numele *Hello*.

Linia *using* (în acest caz, *using System.Web.Services*) realizează includerea *assembly*-urilor necesare (în cazul nostru, a claselor privitoare la serviciile Web).

Linia *[WebService(Namespace="urn:Hello")]* este opțională, dar ne permite să specificăm diferite atribute atașate serviciului Web dezvoltat. În acest caz, am stabilit un spațiu de nume explicit (altfel .NET l-ar fi ales pe cel implicit, desemnat de *http://tempuri.org/*).

[WebMethod] definește faptul că metoda clasei *Hello* face parte din cadrul serviciului Web, fiind disponibilă posibililor clienți. De asemenea, pot fi specificate anumite proprietați legate de operațiile pe care le realizează serviciul.

După ce am creat un astfel de fișier, atunci când el este solicitat de un client via o cerere SOAP transportată prin HTTP, modulul .NET *runtime* va compila codul serviciului (dacă nu a făcut-o deja), pentru a se putea invoca una dintre metodele dorite ale serviciului.

În fapt, .NET permite realizarea mai multor tipuri de cereri:

– cereri privitoare la interfața serviciului (ceea ce oferă acesta), generându-se automat o descriere WSDL a acestuia;

– cereri care se interesează în mod explicit de o metodă furnizată de serviciul în cauză;
– cereri de invocare a unei operații (metode) furnizate de serviciul Web (via SOAP 1.1 sau SOAP 2.0).

Fișierul *salutari.asmx* va fi mutat în directorul public al serverului Web (pentru IIS, în mod implicit este *c:\inetpub\wwwroot*), iar invocarea se va realiza via URI-ul *http://localhost/salutari.asmx* (am presupus că IIS rulează pe mașina locală la portul 80). Un rezultat posibil e cel ilustrat de captura-ecran următoare:

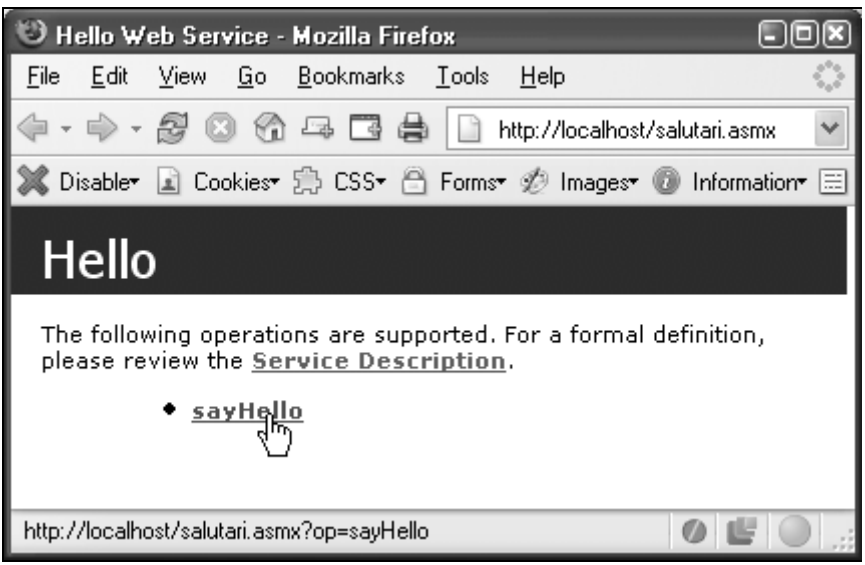

Hello Web Service - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/salutari.asmx
Disable Cookies CSS Forms Images Information
Hello
The following operations are supported. For a formal definition, please review the Service Description.
sayHello
http://localhost/salutari.asmx?op=sayHello

Figura 4. *Interfața Web de interacțiune cu serviciul Web via programul de navigare*

Dând clic pe *sayHello*, vom putea parcurge o descriere detaliată privitoare la maniera de invocarea a metodei *sayHello* folosind cereri SOAP 1.1, SOAP 1.2 sau POST (vezi figura 5).



Figura 5. *Furnizarea informațiilor privitoare la forma cererii SOAP de invocare a metodei serviciului Web*

Pentru a testa funcționalitatea serviciul Web creat, introducem un nume în formularul Web disponibil în cadrul paginii și apăsăm butonul *Invoke*.

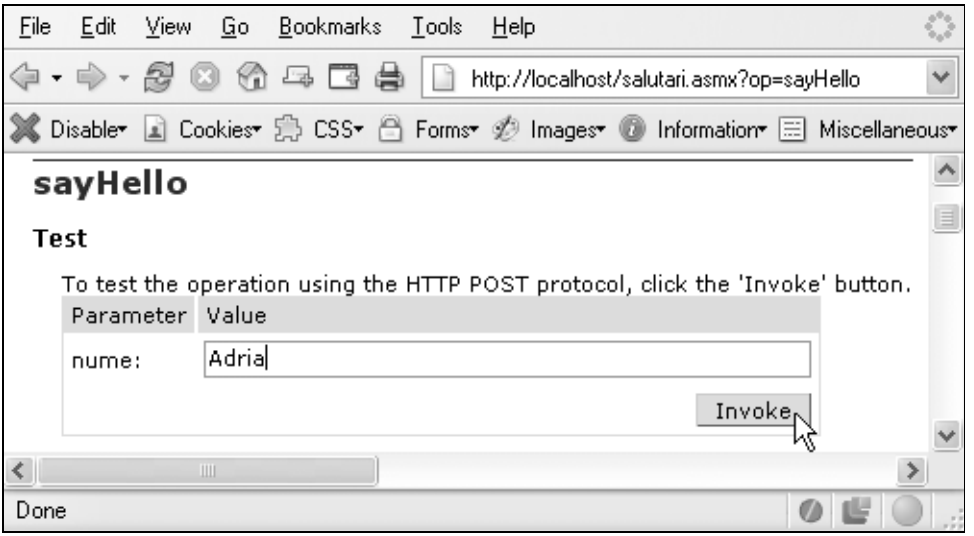


Figura 6. *Invocarea metodei* sayHello

În urma rulării codului, vom obține un rezultat precum cel din figura de mai jos (răspunsul întors este un document XML, așa cum era și de așteptat):

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="urn:Hello">Buna Adria</string>
```

Figura 7. *Rezultatul obținut după execuția metodei invocate*

În cele ce urmează, vom implementa un client simplu care să invoce serviciul Web de mai sus din cadrul unui program C# interactiv. Fragmentul de cod care se ocupă cu invocarea propriu-zisă este următorul:

```
using System;
using System.Diagnostics;
using System.Xml.Serialization;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace Salutari {
[System.Web.Services.WebServiceBindingAttribute(
  Name="HelloSoap", Namespace="urn:Hello")]
public class ClientSalut :
  System.Web.Services.Protocols.SoapHttpClientProtocol
{

public ClientSalut () { // constructor
  // stabilim adresa serviciului Web
```

```
  this.Url="http://localhost/salutari.asmx";
}

[System.Web.Services.Protocols.
   SoapDocumentMethodAttribute("urn:Hello/sayHello",
   RequestNamespace="urn:Hello",
   ResponseNamespace="urn:Hello",
   Use=System.Web.Services.Description.
        SoapBindingUse.Literal,
   ParameterStyle=
        System.Web.Services.Protocols.
        SoapParameterStyle.Wrapped)]
public string sayHello (string nume) {
  // rezultatul obținut (tablou de obiecte)
  // în urma invocării metodei serviciului Web
  object[] rezult =
     this.Invoke ("sayHello", new object[]{nume});
  // întoarcem primul element
  return ((string)(rezult [0]));
}
}
}
```

Construcția *[System.Web.Services.WebServiceBindingAttribute…]* stabilește la compilare ca *assembly*-ul rezultat să fie folosit pentru a invoca un serviciu Web. Când acest *assembly* este compilat, .NET pune la dispoziție un mecanism intern facilitând funcționarea cererilor SOAP.

Linia *System.Web.Services.Protocols.SoapHttpClientProtocol* specifică protocolul care se dorește a fi utilizat (în acest caz, SOAP peste HTTP).

În cadrul clasei s-a mai creat un *proxy* pentru operația *sayHello* și s-au specificat diferite atribute privitoare la serviciul Web invocat. Detalii sunt disponibile în cadrul documentațiilor ce însoțesc *.NET Framework*.

Programul-client recurge la *Windows Forms* și la rulare, în cazul unei compilări cu succes, va afișa o fereastră în care se solicită utilizatorului să introducă un nume. În urma apăsării butonului *Invoke*, vom obține un rezultat precum cel din figura 8.

Când a fost apăsat butonul *Invoke*, s-a produs un eveniment tratat într-o metodă implementată de client. În cadrul acestei metode se apelează *sayHello* din clasa *proxy* care invocă respectiva operație de pe server, returnîndu-se rezultatul de mai sus.

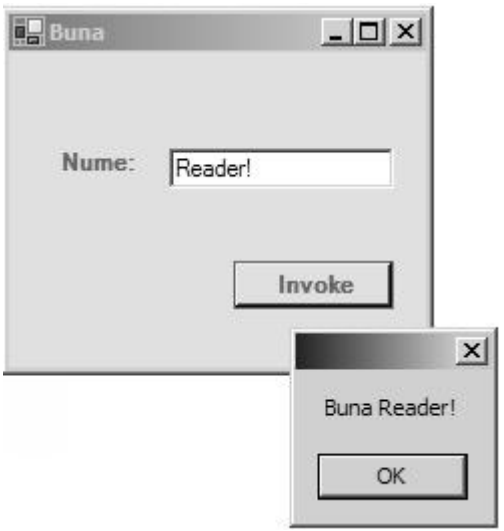


Figura 8. *Rezultatul invocării serviciului Web dintr-un client interactiv*

Invocarea de servicii Web dezvoltate în C# de către clienți concepuți în alte limbaje de programare

Ne propunem în continuare să demonstrăm inter-operabilitatea între serviciile Web create sub Windows – folosind mediul *.NET Framework* – și diverși clienți implementați în C#, Perl și PHP, rulând în Linux și/sau Windows.

Pentru început vom dezvolta în C# un serviciu Web folosit pentru validarea, conform unei scheme XML, a documentelor XML existente pe server.

Editarea codului-sursă și exploatarea serviciului se vor realiza prin intermediul aplicației, disponibile gratuit, *WebMatrix*. Există două variante, una pentru versiunea 1.1 și cealaltă pentru 2.0. *WebMatrix* include și un miniserver Web, permițând verificarea funcționalității serviciilor dezvoltate. De asemenea, se poate utiliza *Microsoft Visual Web Developer Express* – mediu de dezvoltare disponibil, de asemenea, gratuit (vezi figura 9).

Serviciul va expune două metode:

– *CheckIfExists* – verifică existența documentului XML dorit a fi validat;
– *Validate* – realizează validarea propriu-zisă.

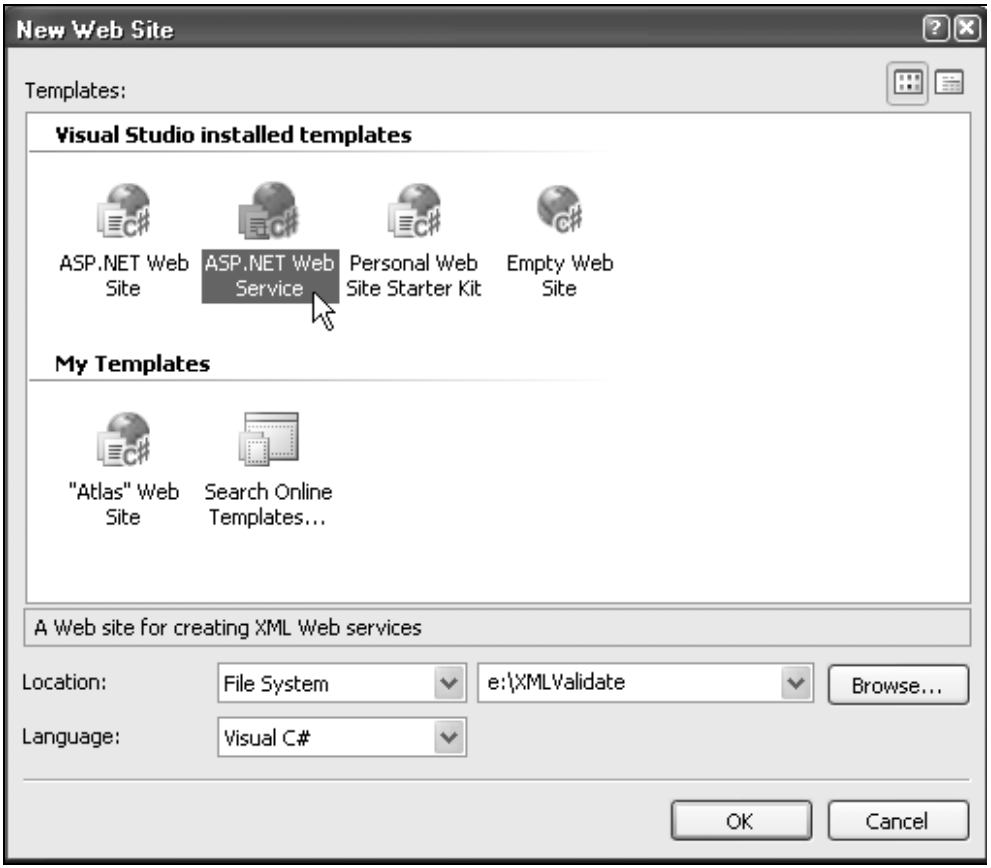


Figura 9. *Crearea unui serviciu Web via șablonul oferit de Microsoft Visual Web Developer Express*

Aceste metode sunt implementate în cadrul clasei purtând numele *XMLValidator*. Această soluție de implementare a fost realizată pentru *.NET Framework* 1.1; în versiunea 2.0, în loc de *XMLValidatingReader* (considerată demodată) se va recurge la *XMLReader*. Lăsăm cititorul să realizeze aceasta ca exercițiu.

```
<%@ WebService language="C#" class="XMLValidator" %>
using System;
using System.Web.Services;
using System.Xml;
using System.Xml.Serialization;
using System.Xml.Schema;
using System.IO;

// clasa care implementează serviciul Web
[WebService(Namespace=
    "http://www.infoiasi.ro/XMLValidator")]
```

```
public class XMLValidator {
     static private int validationMessages;
     static private string result;

     // folosim explicit 'SOAP RPC encoding',
     // pentru fiecare metodă
     [SoapRpcMethod]
     // metoda Web care verifică exist. unui fişier XML
     [WebMethod]
     public bool CheckIfExists (string filename) {
       return File.Exists (filename);
     }
     // metoda Web de validare a unui fişier XML
     [SoapRpcMethod]
     [WebMethod]
     public string Validate (string filename) {
       // cititorul şi validatorul de documente XML
       XmlTextReader tr = null;
       XmlValidatingReader vr = null;
       // iniţial, nici o eroare
       validationMessages = 0;
       result = "";
       // încercăm să procesam documentul XML
       try {
       // instanţiem cititorul de conţinut XML
       tr = new XmlTextReader (filename);
       // instanţiem validatorul XML
       vr = new XmlValidatingReader (tr);
       // tipul implicit de validare
       // foloseşte XML Schema
       vr.ValidationType = ValidationType.Schema;
       // setăm o metodă de semnalare a evenimentelor
       // generate de validatorul XML
       vr.ValidationEventHandler +=
         new ValidationEventHandler (ValidationHandler);
       while ( vr.Read () )
         ; // nu facem nimic, doar citim documentul XML
       } // final de "try"
       catch ( Exception e ) {
           // returnăm mesajul de excepţie
           return result + e.Message;
       }
       finally {
       if ( tr != null )
           tr.Close ();
       if ( vr != null )
           vr.Close ();
       }
```

```
        // întoarcem succes sau numărul de erori
        return (validationMessages == 0 ?
          "valid" : "invalid (" +
                    validationMessages + ")");
      }
      // metodă de tratare a erorilor de validare
      private static void ValidationHandler (
        object sender, ValidationEventArgs args) {
         result += args.Severity +
           "(" + args.Message + "). ";
         validationMessages++;
      }
}
```

Maniera de editare și compilare a codului C# este ilustrată în figura de mai jos.

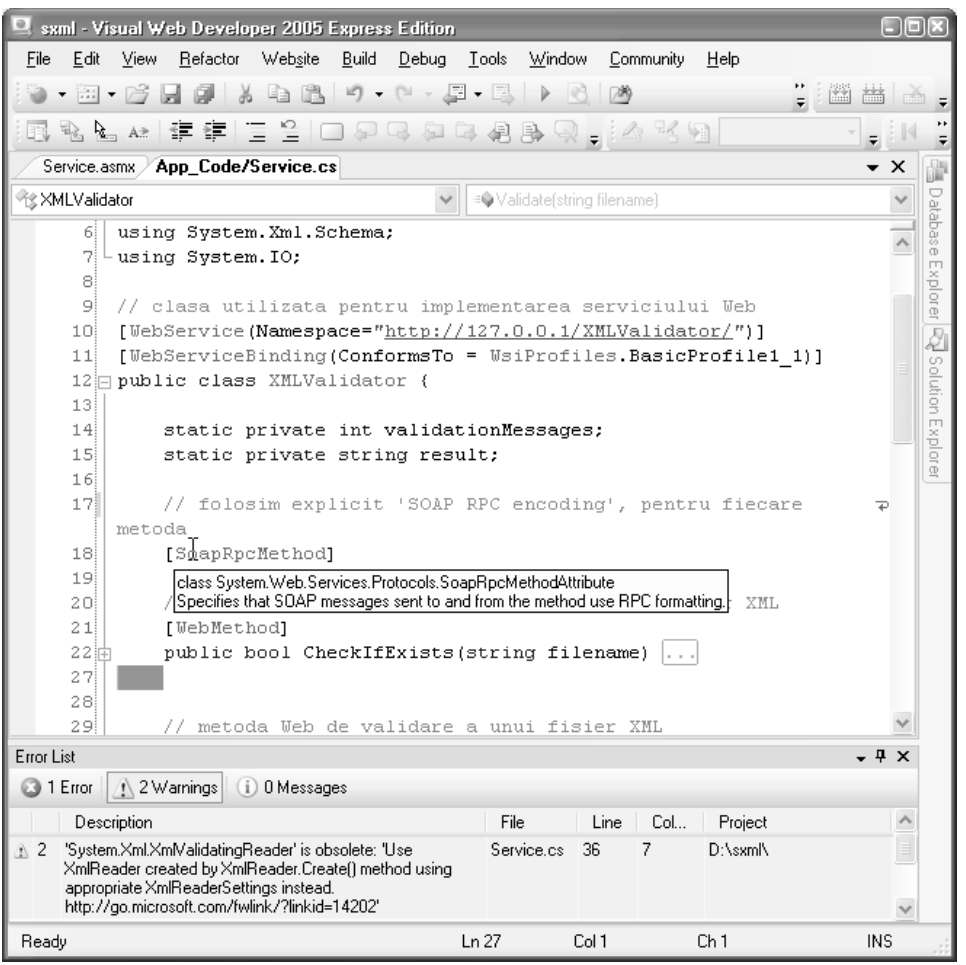

sxml - Visual Web Developer 2005 Express Edition
File Edit View Refactor Website Build Debug Tools Window Community Help
Service.asmx App_Code/Service.cs
XMLValidator
Validate(string filename)
using System.Xml.Schema;
using System.IO;
// clasa utilizata pentru implementarea serviciului Web
[WebService(Namespace="http://127.0.0.1/XMLValidator/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class XMLValidator {
static private int validationMessages;
static private string result;
// folosim explicit 'SOAP RPC encoding', pentru fiecare metoda
[SoapRpcMethod]
class System.Web.Services.Protocols.SoapRpcMethodAttribute
Specifies that SOAP messages sent to and from the method use RPC formatting.
[WebMethod]
public bool CheckIfExists(string filename)
// metoda Web de validare a unui fisier XML
Error List
1 Error 2 Warnings 0 Messages
Description File Line Col... Project
2 'System.Xml.XmlValidatingReader' is obsolete: 'Use XmlReader created by XmlReader.Create() method using appropriate XmlReaderSettings instead. http://go.microsoft.com/fwlink/?linkid=14202' Service.cs 36 7 D:\sxml\
Ready Ln 27 Col 1 Ch 1 INS
Database Explorer
Solution Explorer

Figura 10. *Construirea și depanarea codului serviciului Web în cadrul mediului de dezvoltare Microsoft Visual Web Developer Express*

Mediul .NET pune la dispoziție și o modalitate de generare a descrierii serviciului Web implementat, adică de accesare a reprezentării WSDL asociate. Astfel, putem inspecta comod – cu ajutorul unui instrument precum *WSDL SOAP Analyser*, integrat în deja cunoscutul editor *oXygen*, menționat anterior în cadrul cărții de față – informațiile privitoare la metodele și signaturile acestora (a se vedea figura 11).

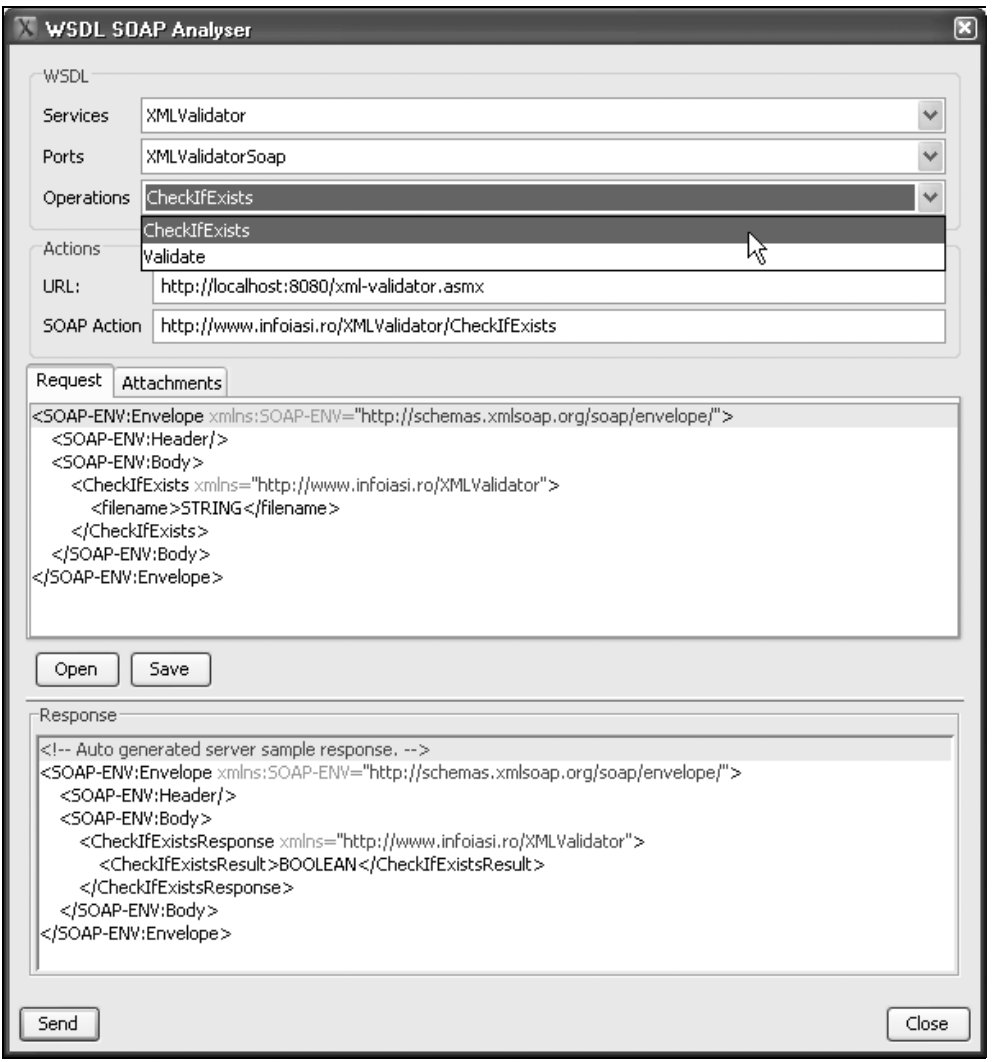


Figura 11. *Obținerea informațiilor privitoare la metodele expuse de un serviciu Web*

Rularea serviciului Web implementat în .NET poate fi realizată atât în cadrul unui server Web precum IIS, cât și în serverul Web oferit de mediul de dezvoltare.

Alte exemple privind dezvoltarea de servicii Web sub platforma .NET sunt disponibile și în volumul lui Sabin Buraga (coord.), *Situri Web la cheie. Soluții profesionale de implementare*, Polirom, Iași, 2004.

2.2. *Un client C# pentru serviciul creat*

O implementare C# minimală a unui client disponibil în linie de comandă, având rolul de invocare a serviciului Web descris mai sus, este următoarea:

```
using System;
class XMLValidatorClient {
   public static void Main(string[] args) {
     if (args.Length == 0) {
     Console.WriteLine("Sintaxa:
           xml-validator-client <document.xml>");
      return;
     }
     try {
        Console.Write("Documentul '{0}' este... ",
          args[0]);
        // creăm o instanță a clasei corespunzătoare
        // serviciului Web
        XMLValidator validator = new XMLValidator();

        // invocăm metoda de verificare a existenței
        // fișierului la nivel de server
        if (validator.CheckIfExists(args[0])) {
          // există, deci putem realiza validarea
          string valid = validator.Validate(args[0]);
          // afișăm ceea ce a întors metoda invocată
          Console.WriteLine(valid + ".");
        }
        else
          Console.WriteLine(
            "inaccesibil (nu există pe server?).");
     } // final de 'try'
     catch (Exception e) {
        // a apărut o excepție
        Console.WriteLine("\nExceptie: " + e.Message);
        return;
     }
   }
}
```

Numele fișierului XML dorit a fi validat este preluat de la linia de comandă, fiind regăsit în șirul de caractere *args[0]*.

Pentru ca programul să funcţioneze, va trebui generată o clasă intermediară (*proxy*), având rol de facilitare a dialogului dintre maşina locală şi serverul pe care va fi rulat serviciul Web. Aceasta se realizează prin comanda următoare, având ca rezultat generarea unui fişier cu numele *XMLValidator.cs*, pe baza descrierii WSDL a serviciul dezvoltat:

```
"c:\Program Files\Microsoft.NET\SDK\v1.1\Bin\wsdl.exe"
http://localhost:8080/xml-validator.asmx?WSDL
```

Calea spre programul *wsdl.exe* (pus la dispoziţie de .NET *Framework*) poate diferi, trebuind să fie ajustată în funcţie de situaţia concretă. Pentru versiunea .NET 2.0, se înlocuieşte „*v1.1*" cu „*v2.0*". În cazul nostru, am presupus că serviciul Web va putea fi invocat pe maşina *localhost*, la portul 8080).

Clientul C# ce validează documentul via serviciul Web de mai sus va trebui compilat printr-o linie de genul:

```
c:\Windows\Microsoft.NET\Framework\v1.1.4322\csc /t:exe /r
:System.Web.dll,System.XML.dll,System.Web.Services.dll
XMLValidator.cs xml-validator-client.cs
```

Interacţiunea dintre un client şi un server Web, via clasa *proxy* generată, este prezentată în figura 12.

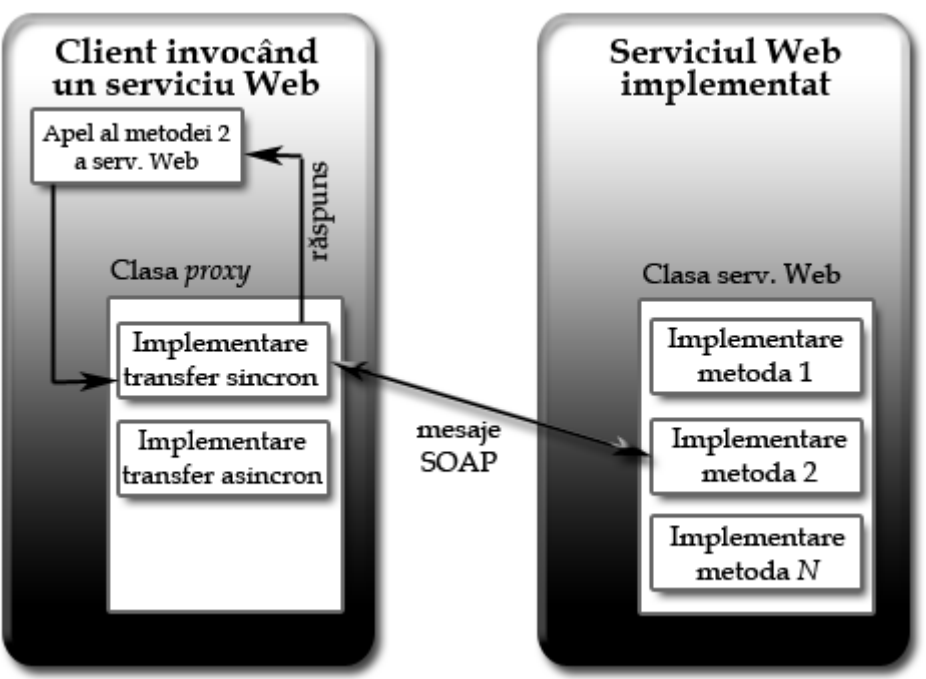


Figura 12. *Interacţiunea dintre un client, clasa* proxy *şi serviciul Web*

Un posibil rezultat este ilustrat de captura-ecran următoare, în care se observă și descrierea WSDL generată automat de *.NET Framework* pentru serviciul Web dezvoltat.

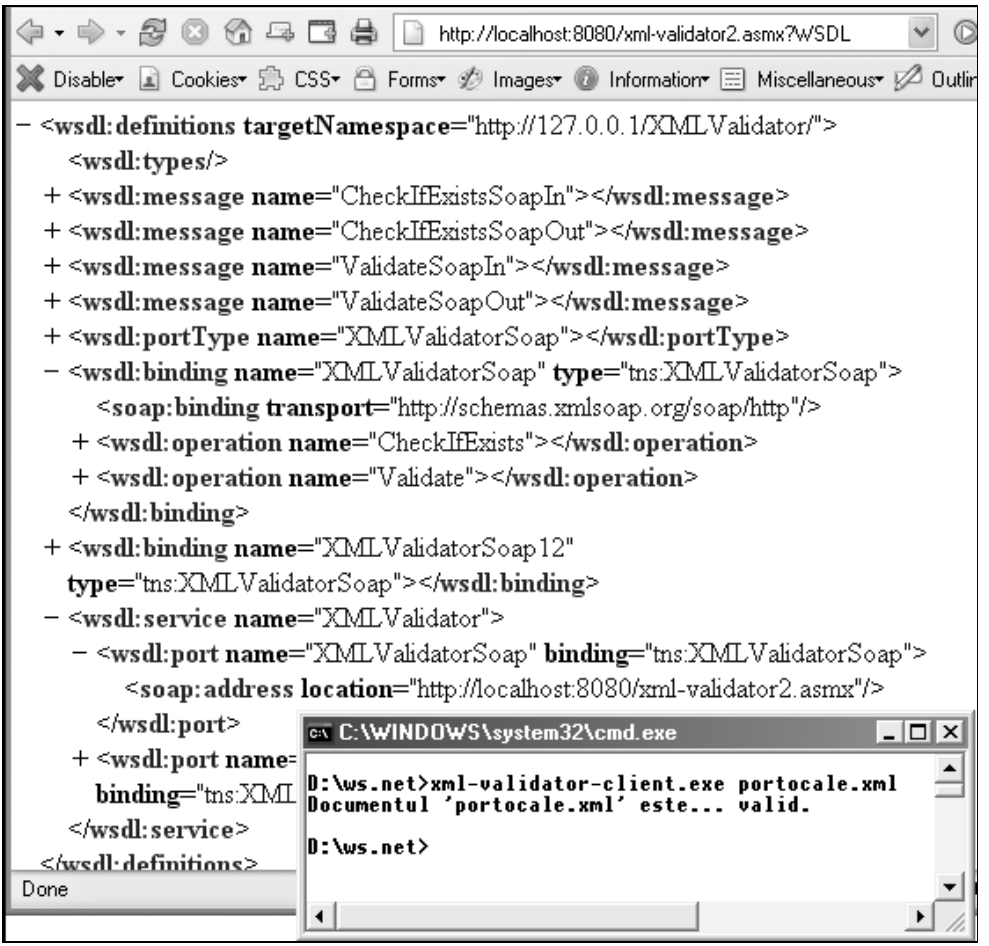


Figura 13. *Răspunsul afișat de clientul care a invocat serviciul Web*

2.3. *Accesarea dintr-un* script *Perl a serviciului Web implementat în .NET*

De asemenea, putem realiza un client conceput în limbajul Perl, apelând la modulul *SOAP::Lite*. Codul acestuia e similar celui descris mai sus, cu excepția faptului că valoarea parametrului *proxy* a constructorului este o adresă de genul *http://winxp.infoiasi.ro:8080/xml-validator.asmx*, iar numele metodelor invocate sunt cele ale serviciului .NET implementat.

Liniile care prezintă interes în contextul de faţă sunt:

```
# ne conectăm la server
my $soap = SOAP::Lite
      ->uri('http://127.0.0.1/XMLValidator/')
                  # URI-ul spaţiului de nume
      ->proxy(
        'http://127.0.0.1:8080/xml-validator2.asmx')
                  # locaţia serviciului
      ->on_action(sub { join('', @_) });

my @params = (SOAP::Data->name('filename', $ARGV[0]));
print "Documentul $ARGV[0] este... ";
# verificăm dacă fişierul XML există pe server
if ($soap->call("CheckIfExists" => @params)->result) {
  # ...şi invocăm metoda de validare
  my $valid =
    $soap->call("Validate" => @params)->result;
  print $valid . ".\n";
}
else {
  print "inaccesibil (nu există pe server?).\n";
}
```

2.4. *Invocarea din PHP a serviciului Web*

Similar, clientul PHP se bazează pe biblioteca *NuSOAP* şi foloseşte descrierea WSDL a serviciului. Instanţierea clientului SOAP în acest caz are forma:

```
// instanţierea clientului SOAP
// pe baza descrierii serviciului Web
$client = new soapclient(
  'http://localhost:8080/xml-validator.asmx?WSDL',
  true);
```

Dacă apelăm *CheckIfExists* cu un nume de fişier inexistent, vom obţine mesaje similare celor din figura 14.

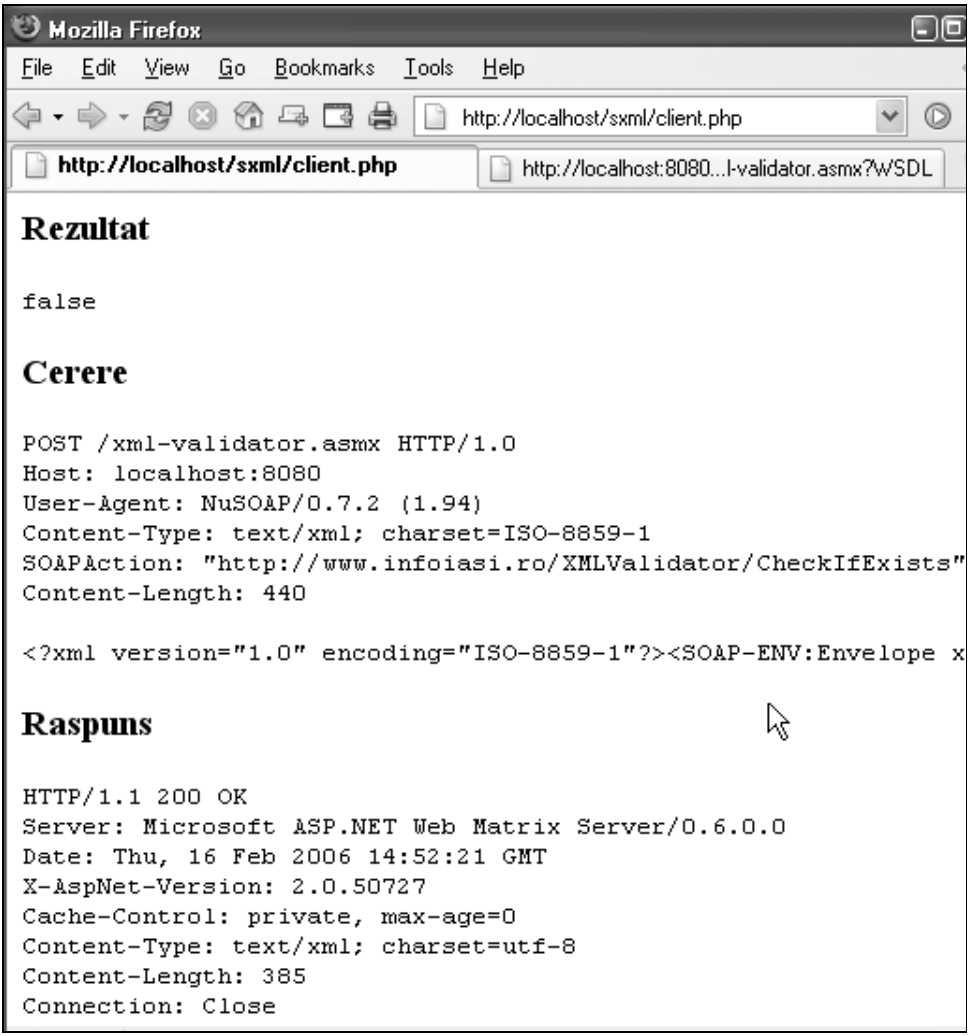


Figura 14. *Rezultatul obținut la invocarea serviciului Web implementat în C# via un client PHP*

Folosind extensia *SOAP* inclusă în PHP 5, vom putea scrie următorul cod-sursă:

```
$doc = $_REQUEST['xml'];
try {
    $client = new SoapClient(
      'http://127.0.0.1:8080/xml-validator2.asmx?WSDL',
      array('trace' => 1));
    // $rez = $client->CheckIfExists ($doc);
    $rez = $client->__call('CheckIfExists',
      array($doc));
```

```
    echo 'Cererea efectuată:<br /><pre class="cod">' .
      htmlspecialchars($client->__getLastRequest(),
        ENT_QUOTES) . '</pre>';
    echo 'Răspunsul obţinut:<br /><pre class="cod">' .
      htmlspecialchars($client->__getLastResponse(),
        ENT_QUOTES) . '</pre>';
    if ($rez == true) {
    $rez = $client->Validate ($doc);
    echo "<p>Documentul $doc este $rez.</p>";
    } else {
    echo "<p>Documentul $doc e inaccesibil
          (nu există pe server?)</p>";
    }
} catch (SOAPFault $exception) {
    echo 'A apărut o excepţie: ' .
      $exception->faultstring;
}
```

Numele documentului dorit a fi validat poate fi furnizat prin GET în şirul de interogare. În invocarea serviciului Web ne folosim de fişierul WSDL asociat acestuia. Astfel, o metodă poate fi apelată direct, ca metodă a obiectului *$client*, instanţă a clasei *SoapClient*. După cum am văzut anterior, o apelare de „nivel scăzut" se poate efectua via metoda *__call()*. În exemplul de faţă am afişat şi cererea şi răspunsul SOAP, aşa cum se remarcă în figura următoare.



Figura 15. *Rezultatul obținut la invocarea serviciului Web via un client ce recurge la extensia* Soap *din PHP 5*

2.5. *Invocarea de servicii Web externe*

Dacă până în momentul de față am scris diverși clienți care invocau metode ale unor servicii Web proprii, e timpul să descriem modalitatea de a obține rezultate din partea unor servicii Web externe, dezvoltate de terțe organizații. Nu vom mai

avea acces la codul-sursă al implementării, ci doar la interfața serviciului – oferită de documentul WSDL ce-l descrie.

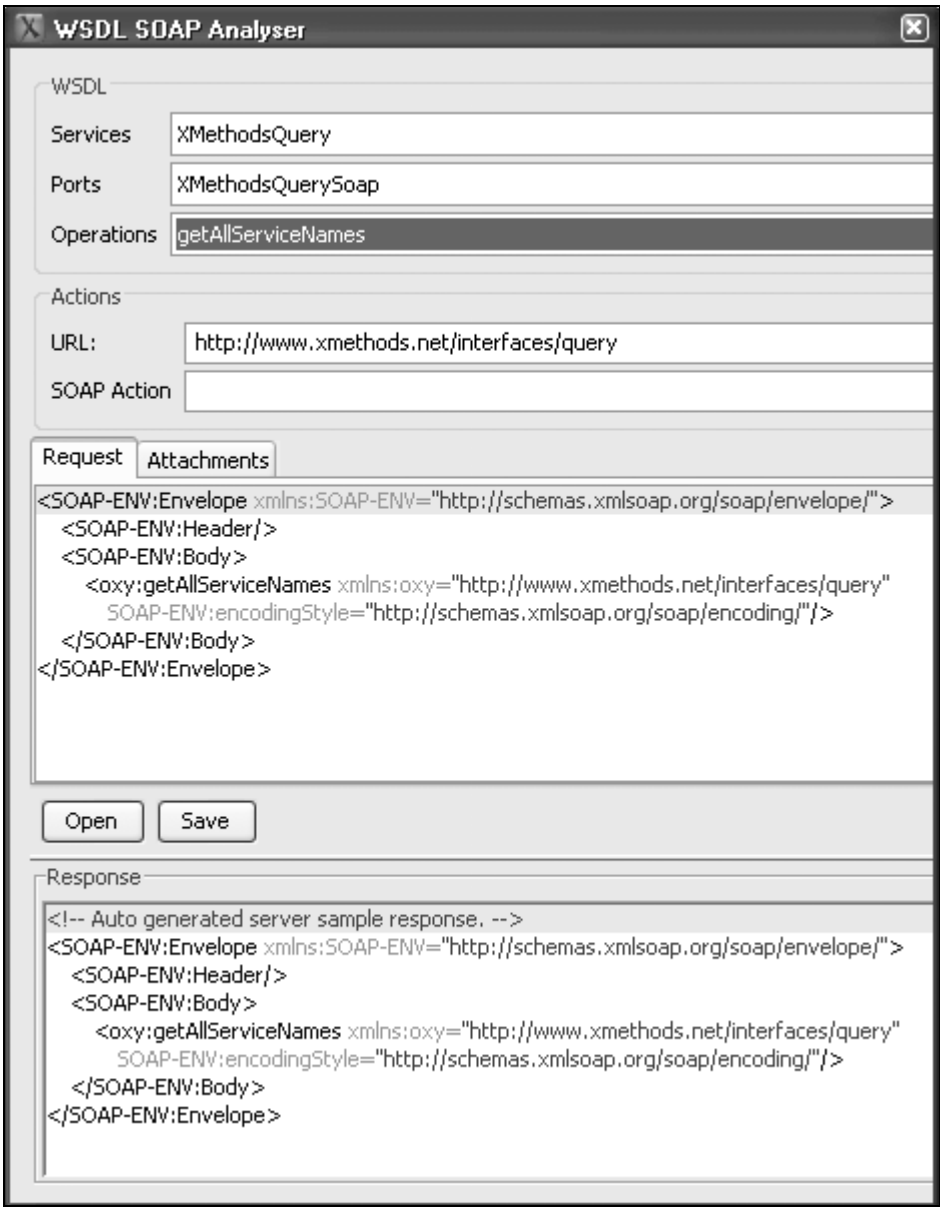


Figura 16. *Consultarea metodei* getAllServiceNames()
a serviciului de interogare disponibil pe situl XMethods

Accesarea serviciului de interogare oferit de XMethods

Unul dintre siturile care pun la dispoziție diverse servicii Web publice este *XMethods – www.xmethods.net*. Pentru a accesa lista serviciilor Web oferite, vom invoca prin SOAP una dintre metodele serviciului Web de interogare oferit de organizația *XMethods*. Consultând documentul WSDL de la adresa *http://www.xmethods.net/interfaces/query.wsdl*, observăm că putem folosi metoda *getAllServicesName()*, care furnizează numele și identificatorii universali unici ai tuturor serviciilor Web înregistrate. Recurgând la instrumentul *WSDL SOAP Analyser* deja amintit în cadrul lucrării de față, obținem informațiile din figura 16.

Clientul care apelează această metodă este scris în limbajul Perl și are următorul cod-sursă (firesc, utilizăm modulul *SOAP::Lite*):

```
use SOAP::Lite;

@params = ( ); # nu există parametri de intrare

# apelăm metoda serviciului Web
$response = SOAP::Lite
   ->proxy("http://www.xmethods.net/interfaces/query")
   ->uri("http://www.xmethods.net/interfaces/query")
   ->call('getAllServiceNames' => @params);

# semnalăm erorile SOAP, dacă există
die "Fault: " . $response->faultcode .
     " " . $response->faultdetail . " " .
     $response->faultstring
  if $response->faultcode;

# afişăm o parte dintre serviciile Web întoarse
foreach $r ($response->dataof ('//name')) {
  if ($r->value =~ /^Co/) { # numele începe cu 'Co'
    print $r->value . "\n";
  }
}
Cererea va avea următoarea formă:
POST http://www.xmethods.net/interfaces/query HTTP/1.1
Accept: text/xml
Accept: multipart/*
Accept: application/soap
Content-Length: 453
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://www.xmethods.net/interfaces/
  query#getAllServiceNames"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc=
    "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   soap:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soap=
    "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <getAllServiceNames
    xmlns="http://www.xmethods.net/interfaces/query"
     xsi:nil="true" />
</soap:Body>
</soap:Envelope>
```

Serverul va întoarce un mesaj precum următorul (observați codificarea SOAP a răspunsului):

```
HTTP/1.1 200 OK
Connection: close
Date: Sat, 12 Aug 2006 09:41:16 GMT
Server: Apache/1.3.26 (Unix) Enhydra-Director/3
        PHP/4.0.6 DAV/1.0.3 AuthNuSphere/1.0.0
Content-Length: 43538
Content-Type: text/xml; charset=UTF-8
Servlet-Engine: Enhydra Application Server/3.1.1b1
  (JSP 1.1; Servlet 2.2; Java 1.4.2_03;
  Linux 2.4.7-10smp i386;
  java.vendor=Sun Microsystems Inc.)

<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:soap=
    'http://schemas.xmlsoap.org/soap/envelope/'
   xmlns:soapenc=
    'http://schemas.xmlsoap.org/soap/encoding/'
   xmlns:n4=
    'http://www.xmethods.net/interfaces/query.xsd'>
<soap:Body soap:encodingStyle=
  'http://schemas.xmlsoap.org/soap/encoding/'
   xmlns:soap=
  'http://schemas.xmlsoap.org/soap/envelope/'>
<n:getAllServiceNamesResponse
  xmlns:n='http://www.xmethods.net/interfaces/query'>
<Result soapenc:arrayType='n4:IDNamePair[476]'>
```

```
    <i>
      <id>uuid:F73777F6-1113-7D28-BE46-2A5E2B8C0313</id>
      <name>Weather by City</name>
    </i>
    <!-- încă 475 de intrări -->
  </Result>
  </n:getAllServiceNamesResponse>
  </soap:Body>
  </soap:Envelope>
```

Se returnează un document XML conținând codificarea unui tablou de perechi (*id*, *name*) pentru fiecare serviciu disponibil. Deoarece sunt foarte multe intrări, programul nostru nu va afișa decât numele serviciilor care încep cu caracterele „*Co*”. Rezultatul rulării este ilustrat în continuare:

```
(infoiasi)$ perl servicii-xmethods.pl
Country Information WebService
Conversions
CountCheat Countdown Service
Congress Member Directory
Code39 Bar Code
ColdFusion Tip-of-the-Day
CountryWebservice
```

Pentru a obține informații referitoare la un anumit serviciu, vom putea recurge la metoda *getServiceDetail()*, care are ca parametru de intrare identificatorul serviciului respectiv și care furnizează un răspuns având structura din figura 17.

Implementarea programului care apelează *getServiceDetail()* folosind Perl sau oricare alt limbaj de programare (lucrurile decurg similar cu cele menționate mai sus) o lăsăm ca exercițiu pentru cititor.

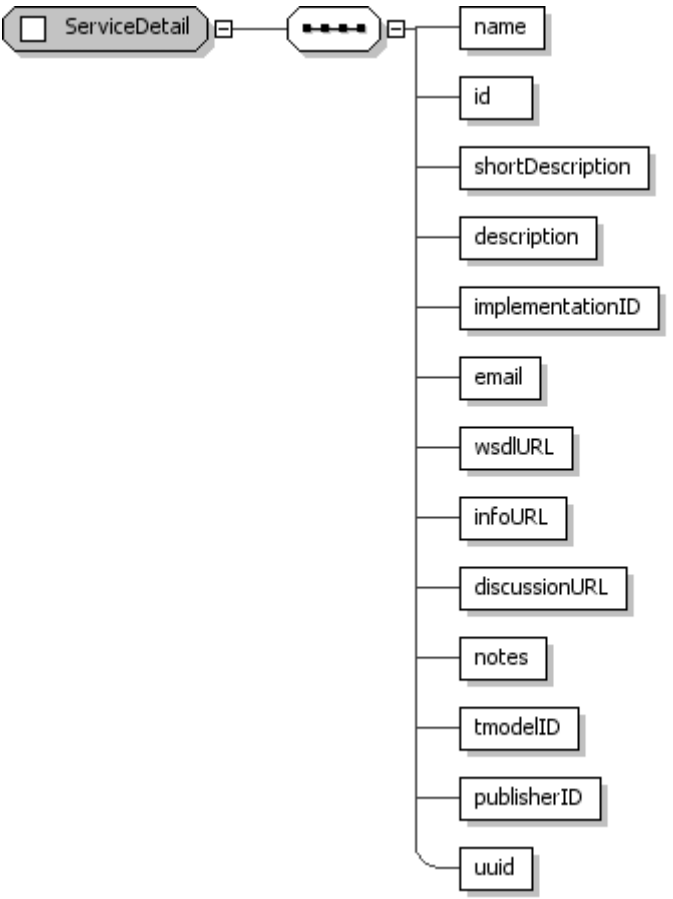


Figura 17. *Structura arborelui corespunzător documentului XML privitor la informațiile despre un serviciu Web oferit de XMethods*

Invocarea serviciilor puse la dispoziție de Google

Motorul de căutare Google pune la dispoziție o serie de servicii Web care pot fi utilizate în cadrul aplicațiilor noastre via SOAP. Vom exemplifica mai jos cum se poate folosi un serviciu de căutare și unul de verificare gramaticală (*spell checking*) dintr-o aplicație scrisă în Visual Basic .NET (adaptare după tutorialele disponibile pe Web).

Primul pas este cel de creare a unui cont la Google și obținerea unei chei cu ajutorul căreia se pot realiza maxim o mie de interogări pe zi – detalii la *http://www.google.com/apis/*.

Următoarea etapă presupune crearea unui proiect de tip *Windows Application* în cadrul mediului de dezvoltare Visual Studio .NET. Vom adăuga o referință la

serviciul oferit de Google, astfel încât să avem acces la acesta (vezi figura 18, în care se pot remarca și metodele ce pot fi folosite).

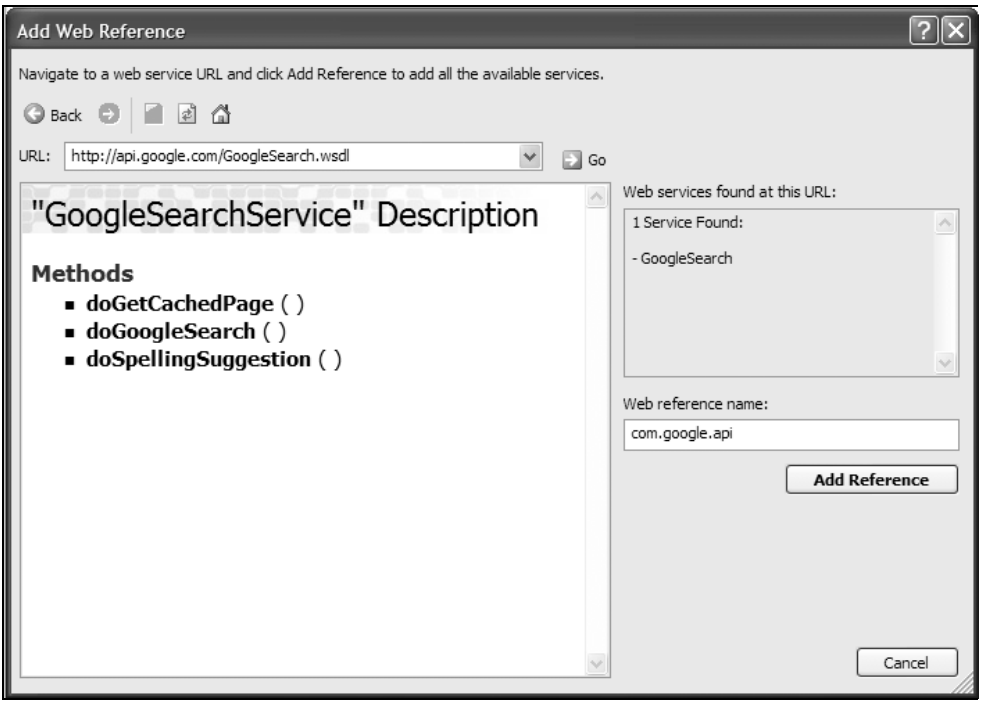
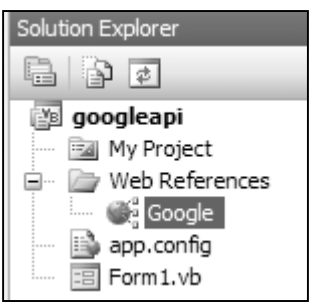


Figura 18. *Adăugarea unei referințe Web spre serviciul Google (se folosește un document WSDL indicat de un URI)*



După introducerea URI-ului *http://api.google.com/GoogleSearch.wsdl*, apăsați butonul *Add Reference*, care determină adăugarea acestei referințe Web la proiectul nostru. În fereastra *Solution Explorer*, referința va fi redenumită drept *Google*; a se urmări figura alăturată.

Vom crea o interfață care va solicita de la utilizator un text folosit ca expresie de căutare, serviciul Google returnând numărul de rezultate (pagini Web) găsite. De asemenea, utilizatorul va putea introduce un șir de caractere scris în limba engleză, iar Google – via metoda *doSpellingSuggestion()* – va furniza termenul corect în urma realizării operației de *spell checking*.

Invocarea primei metode – adică *doGoogleSearch()* – va avea loc la apăsarea butonului etichetat *Numără pag. găsite*, conform codului de mai jos:

```
Private Sub btn_Search_Click (
   ByVal sender As System.Object,
   ByVal e As System.EventArgs) Handles NrPag.Click
   ' Cheia de licenţiere obţinută de la Google
   Dim MyLicenseKey As String
   ' Instanţierea serviciului de căutare
   Dim MyService As Google.GoogleSearchService = New _
     Google.GoogleSearchService()
   ' Rezultatul obţinut în urma invocării
   Dim MyResult As Google.GoogleSearchResult
   MyLicenseKey = "mT2Mpq9QFHLchbCY2F6WYAvEY39I7mDO"
   ' Invocăm metoda
   MyResult = MyService.doGoogleSearch(MyLicenseKey, _
     ExpresiaCautata.Text, 0, 1, False, "", _
     False, "", "", "")
   ' Obţinem nr. de rezultate
   NrRez.Text = "Numarul de rezultate găsite este:" & _
     CStr(MyResult.estimatedTotalResultsCount)
End Sub
```

Liniile responsabile cu apelarea celei de-a doua metode la apăsarea butonului *Spell checking* sunt următoarele:

```
Private Sub btn_SpellChecking_Click (
   ByVal sender As System.Object,
   ByVal e As System.EventArgs) Handles Spelling.Click
   Dim MyLicenseKey As String
   Dim MyService As Google.GoogleSearchService = New _
     Google.GoogleSearchService()
   Dim MyResult As String
   MyLicenseKey = "mT2Mpq9QFHLchbCY2F6WYAvEY39I7mDO"
   MyResult = MyService.doSpellingSuggestion
     (MyLicenseKey, TextulDeVerificat.Text)
   TextCorect.Text = "Google afirmă că e corect: " & _
     MyResult
End Sub
```

Un rezultat posibil e ilustrat de figura 19, în care se pot observa atât rezultatele obţinute printr-o interogare manuală a motorului Google, cât şi cele întoarse de serviciul Web pus la dispoziţie.

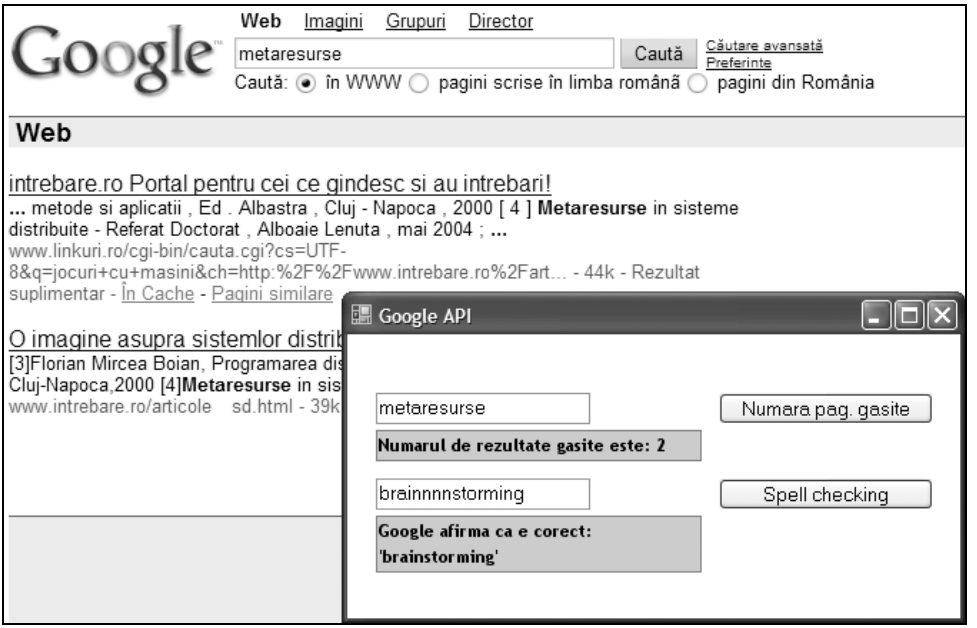

Google
Web Imagini Grupuri Director
metaresurse
Caută
Căutare avansată
Preferinte
Caută: în WWW pagini scrise în limba română pagini din România
Web
intrebare.ro Portal pentru cei ce gindesc si au intrebari!
... metode si aplicatii , Ed . Albastra , Cluj - Napoca , 2000 [4] Metaresurse in sisteme distribuite - Referat Doctorat , Alboaie Lenuta , mai 2004 ; ...
www.linkuri.ro/cgi-bin/cauta.cgi?cs=UTF-8&q=jocuri+cu+masini&ch=http:%2F%2Fwww.intrebare.ro%2Fart... - 44k - Rezultat suplimentar - În Cache - Pagini similare
O imagine asupra sistemlor distrib
[3]Florian Mircea Boian, Programarea dis
Cluj-Napoca,2000 [4]Metaresurse in sis
www.intrebare.ro/articole sd.html - 39k
Google API
metaresurse
Numara pag. gasite
Numarul de rezultate gasite este: 2
brainnnnstorming
Spell checking
Google afirma ca e corect:
'brainstorming'

Figura 19. *Rezultatele obținute via serviciul Web oferit de Google*

Observăm, așadar, cu câtă ușurință se pot încorpora serviciile Web oferite de Google (precum cel de căutare sau cel privitor la informații topografice – via *Google Maps*), în aplicațiile noastre. Utilitatea acestora poate fi foarte mare în crearea de aplicații informative sau de divertisment, care pot utiliza diverse resurse disponibile pe Web. De asemenea, serviciul de *spell checking* se poate integra în cadrul altor programe sau aplicații Web. Alte utilizări sunt cele privitoare la implementarea unor modalități de căutare pe Web din linia de comandă sau *widget*-uri grafice, ori realizarea de statistici privind informațiile despre un anumit subiect de-a lungul timpului.

2.6. *Folosirea instrumentului* gSOAP *pentru C/C++*

Suita de unelte *gSOAP* are drept scop simplificarea dezvoltării de servicii Web prin SOAP și de implementare de clienți pentru invocarea acestora în limbajele C și C++. Se pune la dispoziție un compilator cu rolul de generare a codului-sursă C/C++, pe baza unui fișier antet. De asemenea, avem la dispoziție și o bibliotecă de clase, plus diverse rutine menite să codifice tipurile de date native C++ în XML. Suplimentar, *gSOAP* furnizează și un mecanism de *garbage collection*. Modelul de dezvoltare a serviciilor Web adoptat de *gSOAP* este cel clasic, bazat pe paradigma RPC.

Argumentul că *gSOAP* este un instrument matur este dat de faptul că IBM foloseşte *gSOAP* pentru dezvoltarea de servicii Web destinate dispozitivelor mobile (probabil, datorită necesităţii de utilizare minimă a memoriei şi a unei viteze mari de procesare), iar proiectul european *GridLab* l-a adoptat pentru construirea de servicii destinate sistemelor de tip *grid* (detalii şi în capitolul 7). În acest moment, *gSOAP* este dezvoltat în cadrul *Source Forge*, pornindu-se de la versiunea iniţială oferită de Universitatea din Florida.

Documentaţiile necesare şi o suită de exemple sunt disponibile în cadrul distribuţiei *open source* a pachetului *gSOAP*.

Vom detalia, ca exemplificare, maniera de dezvoltare a unui serviciu Web care, primind un număr, va verifica dacă acesta este prim. Vom crea trei fişiere:

– *nrprim.h*, fişier antet cu informaţii privind funcţiile RPC ce urmează a fi invocate;
– *nrprimserver.c*, reprezentând codul serverului;
– *nrprimclient.c*, reprezentând programul client.

Ca şi în situaţiile de mai sus, vom asocia un spaţiu de nume XML serviciului Web dezvoltat. În fişierul antet se va face corespondenţa între spaţiul de nume stabilit şi prototipurile fiecăreia dintre funcţiile serviciului Web implementat. Codul acestui fişier antet este următorul:

```
//numele serviciului:      nrprim
//stilul de vehiculare a mesajelor: rpc
//spaţiul de nume: http://localhost/nrprim.wsdl
//URI-ul serviciului: http://localhost/nrprimserver.cgi
/* funcţia oferită de serviciul Web */
int ns__nrprim (double numar, double *rezultat);
```

În cazul *gSOAP*, se foloseşte convenţia următoare: toţi parametrii funcţiilor sunt consideraţi parametri de intrare, cu excepţia ultimului (care va conţine rezultatul). De asemenea, valoarea întoarsă este întotdeauna *int*, putând fi folosită pentru diagnosticarea eventualelor erori survenite.

Furnizăm mai jos codul C al serverului:

```
#include "soapH.h"
#include "nrprim.nsmap"

int main(int argc, char **argv) {
  int m, s;          /* descriptori de socket */
  struct soap soap; /* include mesajele SOAP */
  soap_init(&soap); /* iniţializăm... */
```

```
  if (argc < 2)      /* oferim servicii ca un CGI */
    soap_serve(&soap);
  else {
    /* ataşăm serverul la un port
       specificat în linia de comandă (argv[1]) */
    m = soap_bind(&soap, NULL, atoi(argv[1]), 100);
    if (m < 0) {
      /* afişăm eroarea survenită */
      soap_print_fault(&soap, stderr);
      exit(-1);
    }
    fprintf(stderr, "Conexiune reuşită.\n");
    while (1) {
      // acceptăm cereri din partea clienţilor
      s = soap_accept(&soap);
      if (s < 0) {  /* a intervenit o eroare */
        soap_print_fault(&soap, stderr);
        exit(-1);
      }
      fprintf(stderr, "S-a conectat un client.\n");
      soap_serve(&soap);  /* deservim clientul... */
      soap_end(&soap);    /* am terminat */
    }
  }
  return 0;
}

/* funcţia invocată la apariţia cererii
   din partea unui client */
int ns__nrprim (struct soap *soap,
   double numar, double *rezultat) {
  int contor = 2;
  int temp = (int) numar;
  while (contor < temp) { /* verif. dacă e prim */
    if (temp % contor != 0)
      contor++;
    else {
      *rezultat = 0;
      break;
    }
  }
  if (contor == temp)
    *rezultat = 1;
  return SOAP_OK;
}
```

Mai trebuie să descriem codul-sursă al clientului:

```
#include "soapH.h"
#include "nrprim.nsmap"

/* URI-ul serviciului */
const char server[] =
  "http://127.0.0.1/cgi-bin/nrprimserver.cgi";

int main(int argc, char **argv) {
  struct soap soap;
  double numar;      /* numărul solicitat */
  double rezultat;  /* rezultatul primit */
  if (argc < 2) {
    fprintf(stderr, "Introduceţi un număr.\n");
    exit(0);
  }
  soap_init(&soap); /* iniţializare... */
  numar = atoi(argv[1]);
  /* invocăm funcţia via SOAP */
  soap_call_ns__nrprim (&soap, server, "",
     numar, &rezultat);
  /* verificăm dacă au apărut erori */
  if (soap.error) {
     soap_print_fault (&soap, stderr);
    exit(1);
  }
  else {
    if (rezultat == 0)
       printf("Aţi dat un număr care nu este prim\n");
    else
       printf("Aţi dat un număr prim\n");
  }
  /* finalizăm dialogul cu serverul */
  soap_destroy(&soap);
  soap_end(&soap);
  soap_done(&soap);
  return 0;
}
```

Pentru a facilita compilarea diferitelor servicii Web, distribuţia *gSOAP* pune la dispoziţie un fişier *Makefile* pe care puteţi să-l ajustaţi după dorinţă. În fişierul *Makefile* folosit de noi, se apelează compilatorul *gSOAP* RPC, numit *soapcpp2*, în maniera următoare (opţiunea *–c* este folosită pentru a se obţine diversele fişiere-sursă necesare):

```
(infoiasi)$ ./soapcpp2 -c nrprim.h
```

Similar manierei de dezvoltare a aplicațiilor RPC clasice, vor rezulta o serie de fișiere: *soapStub.h*, *soapH.h*, *soapC.c*, *soapClient.c*, *soapServer.c* etc.

În acest moment, codul va fi compilat și legat folosindu-se *stdsoap2.c* (respectiv, *stdsoap2.cpp* pentru C++ în loc de C). Eventual, executabilul rezultat va putea fi apoi instalat ca un *script* CGI.

Să urmărim în figura de mai jos testarea funcționării serviciului:

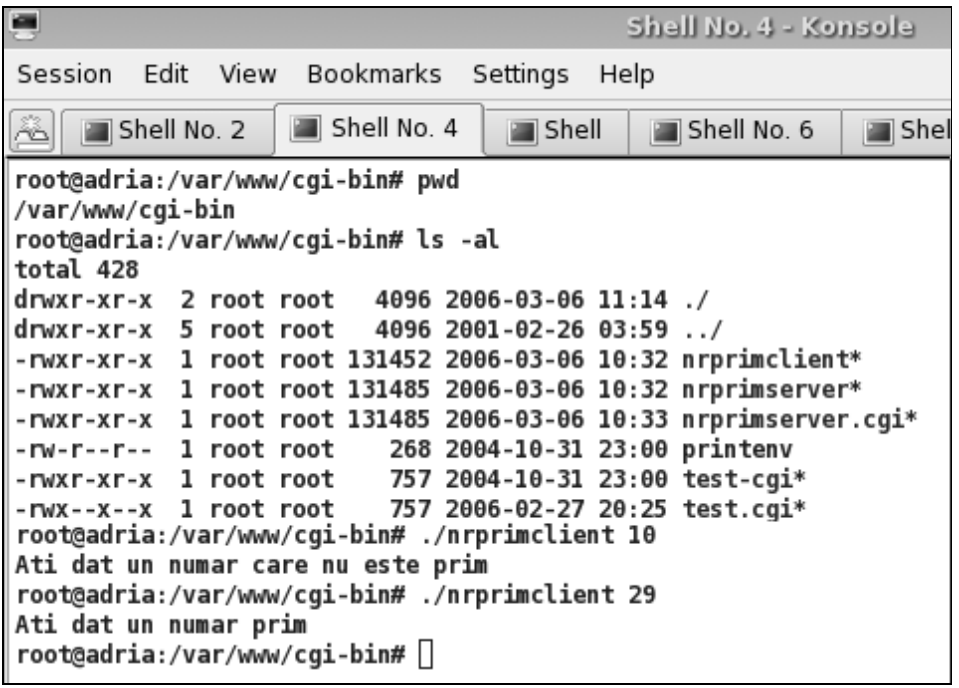

Shell No. 4 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell
Shell No. 6
root@adria:/var/www/cgi-bin# pwd
/var/www/cgi-bin
root@adria:/var/www/cgi-bin# ls -al
total 428
drwxr-xr-x 2 root root 4096 2006-03-06 11:14 ./
drwxr-xr-x 5 root root 4096 2001-02-26 03:59 ../
-rwxr-xr-x 1 root root 131452 2006-03-06 10:32 nrprimclient*
-rwxr-xr-x 1 root root 131485 2006-03-06 10:32 nrprimserver*
-rwxr-xr-x 1 root root 131485 2006-03-06 10:33 nrprimserver.cgi*
-rw-r--r-- 1 root root 268 2004-10-31 23:00 printenv
-rwxr-xr-x 1 root root 757 2004-10-31 23:00 test-cgi*
-rwx--x--x 1 root root 757 2006-02-27 20:25 test.cgi*
root@adria:/var/www/cgi-bin# ./nrprimclient 10
Ati dat un numar care nu este prim
root@adria:/var/www/cgi-bin# ./nrprimclient 29
Ati dat un numar prim
root@adria:/var/www/cgi-bin#

Figura 20. *Rularea programului-client dezvoltat cu* gSOAP

2.7. *Implementarea serviciilor Web în Java*

Instrumentul Apache Axis2

Apache Axis își are originea în *Apache SOAP*, prima soluție, bazată pe SOAP 1.1, oferită de fundația Apache pentru conceperea de servicii Web. Limbajele de programare care pot fi folosite sunt C++ și Java.

Cel mai recent efort, numit *Axis2*, reproiectează arhitectura Axis/Java și Axis/C++, procesarea realizându-se în stilul *stateless* (aceasta permite codului să fie rulat în paralel, de fire de execuție separate). De asemenea, se oferă și un model informațional (*information model*) persistent, sistemul putând fi repornit fără pierderi de date. Arhitectura internă a Apache Axis2 este una modulară,

constând din diverse componente, precum modulul de procesare XML (bazat pe *Apache Xerces*), procesorul de mesaje SOAP, modulul de exploatare a serviciilor (*deployment*), generatorul de documente WSDL, interfața de programare a clienților SOAP și modulul de transport al datelor.

Suplimentar, *Apache Axis2* permite includerea de servicii Web „din zbor", astfel încât servicii Web noi să poată fi rulate fără întreruperea execuției sistemului. De asemenea, se pot realiza actualizări (modificări) ale unui serviciu deja existent, fără a opri sistemul (desigur, eventualele schimbări produse pot conduce la pierderea de informații).

Pentru a putea scrie o aplicație în Java folosind *Axis2* aveți nevoie de următoarele componente software:

– *Apache Geronimo* – exemplele furnizate mai jos au fost rulate pe acest server, pe baza căruia este construit și *IBM WebSphere Community Edition*. Desigur, puteți recurge și la un alt server. Pentru a prelua *Geronimo*, vizitați adresa *http://geronimo.apache.org/downloads.html*.
– *Apache Axis2* (ori o altă implementare SOAP). Instrumentele puse la dispoziție reduc timpul de dezvoltare a codului. Puteți obține Apache Axis2 de la adresa *http://ws.apache.org/axis2/download.cgi* (remarcăm faptul că distribuția *.war* este folosită pentru dezvoltarea de servicii Web, iar cea binară vă ajută la crearea clienților);
– *Java 2 Standard Edition* începând cu versiunea 1.4.

Pornind de la premisa că uneori instalarea diferitelor instrumente „costă" mai mult timp decât dezvoltarea propriu-zisă a aplicației (deși nu este cazul aici), vom prezenta în continuare pașii necesari realizării unui serviciu Web în Java, folosind componentele software enumerate mai sus.

După ce ați transferat *Geronimo*, trebuie doar să dezarhivați serverul în oricare director doriți (pentru prevenirea eventualelor probleme, evitați instalarea – în cazul Windows – în directoarele *Program Files* sau *Documents and Settings*). Pentru exemplele următoare, presupunem serverul Geronimo stocat în directorul *c:\geronimo*.

Puteți apoi porni serverul, fie din linia de comandă, fie rulând fișierul *startup.bat*. În primul caz, introduceți de la *prompt*-ul sistemului liniile:

```
cd \geronimo\bin
java -jar server.jar
```

Testați rularea serverului, introducând adresa *http://localhost:8080* în browser-ul preferat. În caz de succes, veți obține informațiile din figura de mai jos.

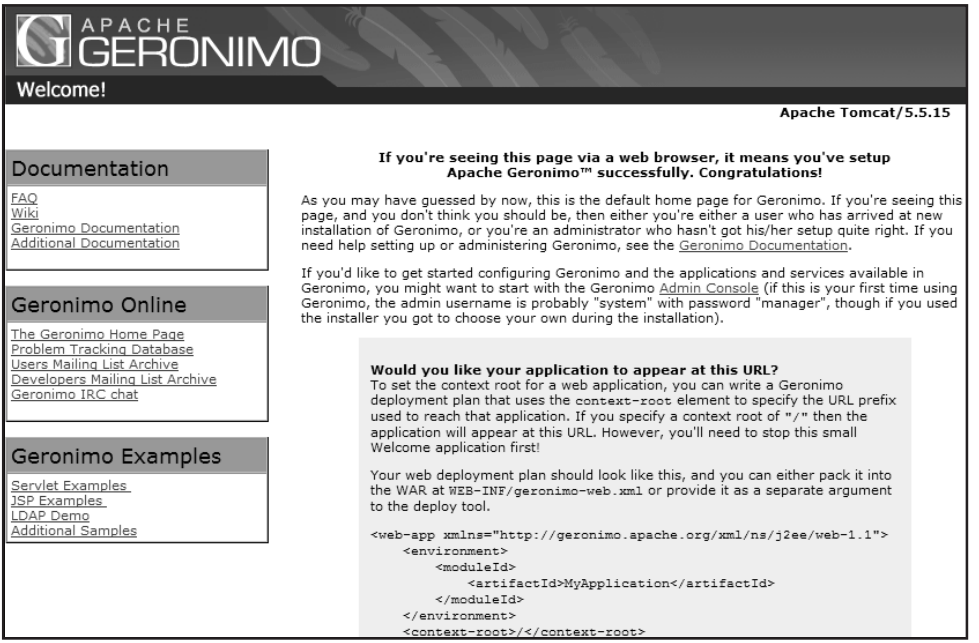

APACHE GERONIMO
Welcome!
Apache Tomcat/5.5.15
Documentation
FAQ
Wiki
Geronimo Documentation
Additional Documentation
Geronimo Online
The Geronimo Home Page
Problem Tracking Database
Users Mailing List Archive
Developers Mailing List Archive
Geronimo IRC chat
Geronimo Examples
Servlet Examples
JSP Examples
LDAP Demo
Additional Samples
If you're seeing this page via a web browser, it means you've setup Apache Geronimo™ successfully. Congratulations!
As you may have guessed by now, this is the default home page for Geronimo. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Geronimo, or you're an administrator who hasn't got his/her setup quite right. If you need help setting up or administering Geronimo, see the Geronimo Documentation.
If you'd like to get started configuring Geronimo and the applications and services available in Geronimo, you might want to start with the Geronimo Admin Console (if this is your first time using Geronimo, the admin username is probably "system" with password "manager", though if you used the installer you got to choose your own during the installation).
Would you like your application to appear at this URL?
To set the context root for a web application, you can write a Geronimo deployment plan that uses the context-root element to specify the URL prefix used to reach that application. If you specify a context root of "/" then the application will appear at this URL. However, you'll need to stop this small Welcome application first!
Your web deployment plan should look like this, and you can either pack it into the WAR at WEB-INF/geronimo-web.xml or provide it as a separate argument to the deploy tool.
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">
<environment>
<moduleId>
<artifactId>MyApplication</artifactId>
</moduleId>
</environment>
<context-root>/</context-root>

Figura 21. *Testarea rulării serverului* Geronimo

Următoarea etapă este instalarea instrumentului *Apache Axis2*. Folosiți consola de administrare *Geronimo* (numele de utilizator și parola în mod implicit sunt „*system*", respectiv „*manager*") și selectați opțiunea *Deploy New*. Testați desfășurarea cu succes a operațiunii, prin vizitarea adresei *http://localhost:8080/axis2/*.

Din acest moment putem crea propriul nostru serviciu Web. Indiferent ce tehnologii folosiți, există două abordari de dezvoltare: fie pornind de la documentul WSDL ce descrie interfața serviciului (*contract first approach*), fie scriind codul-sursă al implementării (*code first approach*). Prima metodă presupune buna cunoaștere a specificațiilor WSDL (a se revedea capitolul 3).

Mai simplu este să concepem o clasă Java și apoi să o transformăm în serviciu Web. În abordarea *Axis2*, după scrierea codului clasei mai urmează să precizăm așa-numitul descriptor al serviciului, memorat în fișierul *services.xml*, și să creăm o arhivă de exploatare a aplicației.

Ne propunem să realizăm un serviciu simplu de salutare a utilizatorului:

```
import org.apache.axis2.context.ServiceContext;
import org.apache.axis2.context.OperationContext;
public class HelloJavaSoap {
  public String echoJavaSoap(String param) {
    return "Echooo... " + param;
  }
}
```

Am scris o clasă Java cu o singură metodă publică. Această metodă primește un parametru de tip *string* și întoarce valoarea parametrului prefixată de un text.

Urmează să creăm documentul *services.xml* care indică mediului cum să configureze și să expună posibililor clienți serviciul Web în cauză:

```
<service name="HelloJavaSoap">
   <description>
      Primul serviciu Web folosind Axis2
   </description>
   <parameter name="ServiceClass" locked="false">
      HelloJavaSoap
   </parameter>
   <operation name="echoJavaSoap">
      <messageReceiver class=
   "org.apache.axis2.rpc.receivers.RPCMessageReceiver"
      />
   </operation>
</service>
```

Pentru a stabili corespondența între serviciu și clasa Java care îl implementează, am adăugat un atribut cu numele *ServiceClass*, a cărui valoare reprezintă calea spre locația unde se află memorată implementarea. Elementul *operation* specifică faptul că metoda oferită de serviciul Web este *echoJavaSoap*, mesajele fiind recepționate via clasa pusă la dispoziție de *Apache Axis2*. După cum se poate observa, se recurge la un transfer în stilul RPC.

Următorul pas este să creăm o arhivă cu extensia *.aar* (*Axis2 Archive*) care include codul compilat (fișierul *.class*) și documentul *services.xml*. Structura internă a acestei arhive – de fapt, un *.zip* cu extensie diferită – are forma:

```
META-INF\
   Services.xml
HelloJavaSoap.class
```

Mai urmează să încărcăm serviciul prin interfața de administrare disponibilă la *http://localhost:8080/axis2/axis2-admin/*, selectând opțiunea *Upload Service* (urmăriți figura 22).

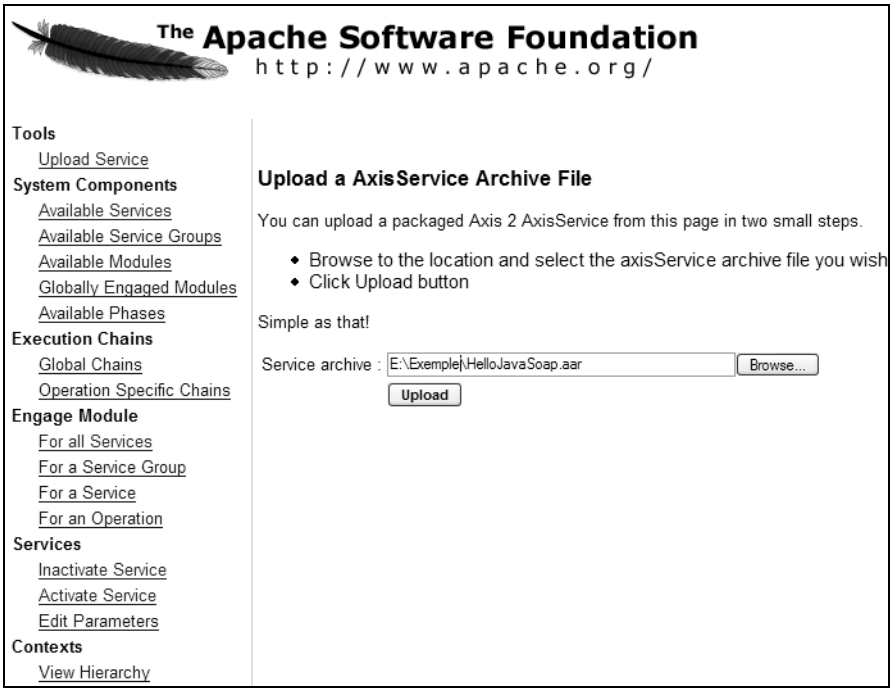


Figura 22. *Încărcarea serviciului Web în vederea invocării ulterioare*

După aceasta, dați clic pe legătura *Available Services* pentru a obține mesajele din captura-ecran următoare:



Figura 23. *Lista serviciilor disponibile (aici, serviciul Web* HelloJavaSoap*)*

În cadrul distribuției *Axis2*, sunt disponibile o serie de exemple pe care le recomandăm spre studiu. Ne oprim asupra unui program, numit *DynamicInvoker*, care reprezintă un client Java pentru invocarea serviciilor Web (cel anterior, serviciile Web externe prezentate în secțiunea 2.5 etc.) pe baza documentului WSDL asociat.

Pentru rularea acestui program, trebuie să stabilim valoarea corespunzătoare pentru variabila de mediu CLASSPATH, astfel încât să cuprindă locația unui procesor XML (*e.g.*, *Apache Xerces*) și toate fișierele *.jar* din directorul *lib*. După compilarea codului, îl rulăm astfel:

```
java samples.client.DynamicInvoker http://localhost:8080/
axis2/services/HelloJavaSoap?wsdl  echoJavaSoap "avem
portocale albastre!"
```

Vom obține următorul răspuns (am omis anumite mesaje de avertisment privitoare la lipsa suportului pentru jurnalizare *log4j*):

```
Reading WSDL document from 'http://localhost:8080/
axis2/services/HelloJavaSoap?wsdl'
Preparing Axis dynamic invocation
Executing operation echoJavaSoap with parameters:
>echoJavaSoap>param0=avem portocale albastre!
>echoJavaSoapResponse>return=Echooo... avem portocale
albastre!

Done!
```

Din cele prezentate mai sus, putem considera că mediul *Axis2* oferă un mecanism flexibil și rapid de dezvoltare a serviciilor Web. Ca alternative, menționăm JWSDP (*Java Web Services Developer Pack*) pus la dispoziție de compania Sun și noua versiune J2SE 1.6, care include suport nativ pentru crearea și invocarea de servicii Web.

Importarea serviciilor Web în Java

Pe baza documentului WSDL ce descrie serviciul implementat în C# (a se revedea secțiunea 2.1), vom putea realiza așa-numita importare a interfeței lui în limbajul Java. Această acțiune poate fi realizată ușor, grație instrumentului *wsimport* pus la dispoziție de Java 2 Standard Edition – versiunea 1.6 (la momentul scrierii acestei cărți, am folosit varianta J2SE 1.6 beta 2). Rulându-l cu următorii parametri dați în linia de comandă, vom obține în cadrul directorului *xml* codul-sursă Java al pachetului *ro.infoiasi.xmlvalidator* ce desemnează interfața serviciului:

```
c:\jdk1.6.0\bin>wsimport http://localhost:8080/
xml-validator.asmx?WSDL -s .\xml -verbose
ro\infoiasi\xmlvalidator\CheckIfExists.java
ro\infoiasi\xmlvalidator\CheckIfExistsResponse.java
ro\infoiasi\xmlvalidator\ObjectFactory.java
ro\infoiasi\xmlvalidator\Validate.java
ro\infoiasi\xmlvalidator\ValidateResponse.java
ro\infoiasi\xmlvalidator\XMLValidator.java
ro\infoiasi\xmlvalidator\XMLValidatorSoap.java
ro\infoiasi\xmlvalidator\package-info.java
```

Fișierul *XMLValidatorSoap.java* conține definițiile metodelor – accesate via SOAP – expuse de serviciul Web și are următorul cod generat automat (din care, de altfel, se poate deduce și maniera de dezvoltare a serviciilor Web în J2SE 1.6):

```
package ro.infoiasi.xmlvalidator;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

// Numele şi spaţiul de nume al serviciului Web
@WebService(name = "XMLValidatorSoap",
  targetNamespace =
    "http://www.infoiasi.ro/XMLValidator")
public interface XMLValidatorSoap {
  // metoda care verifică existenţa fişierului
  @WebMethod(operationName = "CheckIfExists",
      action =
  "http://www.infoiasi.ro/XMLValidator/CheckIfExists")
  @WebResult(name = "CheckIfExistsResult",
      targetNamespace =
       "http://www.infoiasi.ro/XMLValidator")
  // desemnează clasa responsabilă cu încapsularea
  // cererii SOAP
  @RequestWrapper(localName = "CheckIfExists",
      targetNamespace =
       "http://www.infoiasi.ro/XMLValidator",
      className =
       "ro.infoiasi.xmlvalidator.CheckIfExists")
  // desemnează clasa responsabilă cu încapsularea
  // răspunsului SOAP
  @ResponseWrapper(localName = "CheckIfExistsResponse",
      targetNamespace =
       "http://www.infoiasi.ro/XMLValidator",
      className =
       "ro.infoiasi.xmlvalidator.CheckIfExistsResponse")
```

```
  public boolean checkIfExists(
    // meta-date pentru parametrul de intrare
    @WebParam(name = "filename",
              targetNamespace =
               "http://www.infoiasi.ro/XMLValidator")
    String filename);
  /* similar şi pentru a doua metodă */
}
```

Adnotările asociate serviciului Web sunt generate conform documentului WSDL și sunt disponibile în fișierul *XMLValidator.java*:

```
@WebServiceClient(name = "XMLValidator",
  targetNamespace =
    "http://www.infoiasi.ro/XMLValidator",
  wsdlLocation =
    "http://localhost:8080/xml-validator.asmx?WSDL")
public class XMLValidator extends Service {
  private final static URL XMLVALIDATOR_WSDL_LOCATION;
  static {
    URL url = null; // adresa documentului WSDL
    try {
      url = new URL(
      "http://localhost:8080/xml-validator.asmx?WSDL");
    }
    catch (MalformedURLException e) {
      e.printStackTrace();
    }
    XMLVALIDATOR_WSDL_LOCATION = url;
  }
  public XMLValidator(URL wsdlLocation,
    QName serviceName) {
    super(wsdlLocation, serviceName);
  }
  public XMLValidator() {
    super(XMLVALIDATOR_WSDL_LOCATION,
      new QName("http://www.infoiasi.ro/XMLValidator",
                "XMLValidator"));
  }
  // se defineşte punctul terminal al serviciului
  // (întoarce un obiect XMLValidatorSoap,
  //  a cărui clasă a fost dată mai sus)
  @WebEndpoint(name = "XMLValidatorSoap")
  public XMLValidatorSoap getXMLValidatorSoap() {
    return (XMLValidatorSoap)super.getPort(new
      QName("http://www.infoiasi.ro/XMLValidator",
        "XMLValidatorSoap"), XMLValidatorSoap.class);
  }
}
```

Se remarcă existența pachetului *javax.xml.ws* disponibil începând cu J2SE 1.6 și J2EE 1.5, care oferă suportul necesar dezvoltării facile și invocării de servicii Web atât prin SOAP, cât și recurgând la paradigma REST. Acest pachet formează JAX-WS, interfața de programare de bază a serviciilor Web.

Conform JAX-WS, un serviciu Web în rulare reprezintă o instanță (obiect) a unei clase Java, adnotat prin *javax.jws.WebService*. Interfața expusă către posibilii clienți este o interfață Java privitoare la punctul terminal al serviciului (SEI – *Service Endpoint Interface*), care nu trebuie neapărat specificată de programator, deoarece ea poate fi suplinită de clasa care implementează serviciul. După scrierea codului și realizarea compilării, va putea fi creat un fișier *.war* care va fi exploatat în cadrul unui server de aplicații precum Tomcat. În timpul invocării, serverul de aplicații va genera automat clasele responsabile cu schimbul de mesaje cu posibilii clienți.

Similar manierei folosite la .NET, cel mai simplu serviciu Web (de salutare a utilizatorului) are codul următor:

```
package salut;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService()
public class Salut { // clasă implementând serviciul
  private String mesaj = new String("Salut, ");

  // o singură metodă
  @WebMethod()
  public String sayHello (String nume) {
    return mesaj + nume;
  }
}
```

Corespunzător, clientul Java care invocă metoda de mai sus este compus din următoarele linii:

```
import javax.xml.ws.WebServiceRef;
import salut.ServiciuSalut;
import salut.Salut;

public class ClientSalut {
  @WebServiceRef(wsdlLocation=
    "http://localhost:8080/salutari/salutari?wsdl")
  static ServiciuSalut serviciu;
```

```
  public static void main(String[] args) {
    try {
      ClientSalut client = new ClientSalut();
      client.invoca(args);
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  // încearcă să invoce serviciul
  public void invoca(String[] args) {
    try {
      System.out.println("Vom invoca " + serviciu);
      Salut port = serviciu.getHelloPort();
      String nume;
      // preluăm numele din linia de comandă
      nume = (args.length > 0) ?
        args[0] : "Portocal Escu";
      String raspuns = port.sayHello(nume);
      System.out.println(raspuns);
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

2.8. *Suportul oferit de mediul Delphi pentru crearea şi utilizarea serviciilor Web*

Dezvoltarea serviciului

La finalul acestui subcapitol vom prezenta maniera de implementare în Delphi (Object Pascal) a unui serviciu Web simplu. Codul-sursă a fost testat sub *Delphi 6 Enterprise Edition*, care oferă suport pentru SOAP 1.1.

Pentru crearea serviciului Web, avem la dispoziţie un asistent (*wizard*) pe care-l putem apela prin intermediul operaţiei *Other* a opţiunii *New* din cadrul meniului *File*. Selectând *tab*-ul *WebServices*, vom da clic pe *Soap Server Application*. A se urmări figura de mai jos.

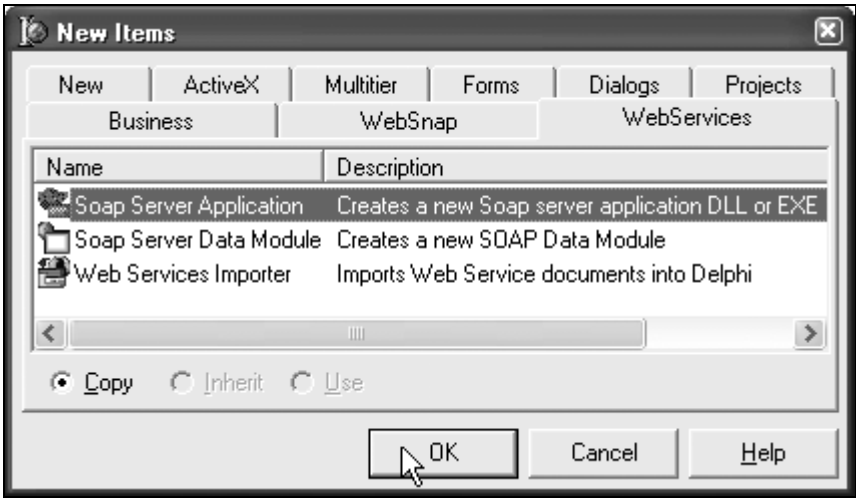


Figura 24. *Apelarea asistentului de creare a unui serviciu Web*

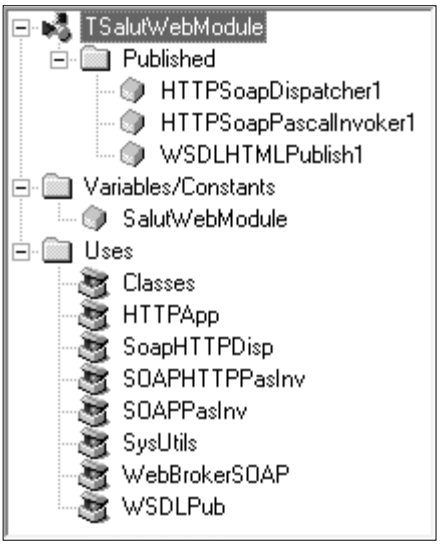
Va apărea o fereastră în care trebuie să alegem tipul de aplicație ce va fi generată (vezi figura 25). Vom opta pentru realizarea unui executabil CGI care va putea fi ușor exploatat ulterior în cadrul serverului Apache. Drept alternative, ni se oferă posibilitatea să implementăm serviciul ca modul Apache ori ca o bibliotecă dinamică executată în cadrul unui server precum IIS.



Figura 25. *Alegerea tipului de aplicație Web care va încapsula serviciul dezvoltat*

Mediul Delphi va genera un *unit*, pe care-l vom salva sub denumirea *SWebMod.pas*, care va îngloba trei componente:

– *HTTPSoapDispatcher1* – dispecerul SOAP peste HTTP, responsabil cu preluarea cererilor şi trimiterea răspunsului;
– *HTTPSoapPascalInvoker1* – clasa responsabilă cu (de)serializarea mesajelor SOAP în obiecte Pascal;
– *WSDLHTMLPublish1* – clasa care generează descrierea WSDL a serviciului.



Acest modul Web are structura ilustrată de figura alăturată, iar codul lui sursă este furnizat în continuare:

```
unit SWebMod;

interface
uses
  SysUtils, Classes, HTTPApp, WSDLPub, SOAPPasInv,
  SOAPHTTPPasInv, SoapHTTPDisp, WebBrokerSOAP;
type
  { clasa responsabilă cu publicarea serviciului }
  TSalutWebModule = class(TWebModule)
    { dispecerul SOAP peste HTTP }
    HTTPSoapDispatcher1: THTTPSoapDispatcher;
    { clasa responsabilă cu (de)serializarea
      mesajelor SOAP în obiecte Pascal }
    HTTPSoapPascalInvoker1: THTTPSoapPascalInvoker;
    { clasa care generează doc. WSDL al serviciului }
    WSDLHTMLPublish1: TWSDLHTMLPublish;
  end;

var
  SalutWebModule: TSalutWebModule;

implementation
{$R *.DFM}
end.
```

Mai urmează să scriem o clasă care va implementa efectiv serviciul Web dorit. Vom crea astfel *unit*-ul *salut.pas*, al cărui cod este următorul:

```
unit salut;

interface
uses InvokeRegistry;
type
  { o interfață invocabilă }
  ISalut = interface(IInvokable)
    ['{D7532EC7-EA7F-4610-ADE5-B3634BB0F343}']
    function Salut: String; stdcall;
    function NumaraZile: Byte; stdcall;
  end;
  { o clasă implementând interfața }
  TSalut = class (TInvokableClass, ISalut)
  public
    { metodele expuse de serviciul Web }
    function Salut: String; stdcall;
    function NumaraZile: Byte; stdcall;
  end;

implementation
uses SysUtils, DateUtils;
{ implementarea metodelor serviciului Web }
function TSalut.Salut: String; stdcall;
begin
  Result := 'Salut din Delphi, prin SOAP!';
end;

function TSalut.NumaraZile: Byte; stdcall;
begin
  Result := DaysBetween (Now,
    EncodeDate(2006, 12, 31));
end;

initialization
  { înregistrarea interfeței și clasei }
  InvRegistry.RegisterInterface(TypeInfo(ISalut));
  InvRegistry.RegisterInvokableClass(TSalut);
end.
```

După cum se observă, specificăm pentru început o interfață invocabilă, având două metode ce vor fi expuse de către serviciul Web de față. Acestei interfețe i se asociază un identificator universal unic, în maniera componentelor COM. De asemenea, oferim și o implementare via o clasă invocabilă. Sunt definite două

funcţii, una care întoarce un mesaj (şir de caractere) de salut, cealaltă oferind numărul de zile (un întreg de tip *Byte*) rămase din momentul execuţiei până la finalul anului 2006.

Cele două *unit*-uri le includem în proiectul cu numele *SalutWS.dpr* şi generăm fişierul executabil. Pentru a putea fi efectiv utilizat, acesta din urmă trebuie plasat în directorul *htdocs* al serverului Apache pentru Windows (în cazul nostru, folosim distribuţia *Apache2Triad*). Mai trebuie să configurăm serverul să trateze fişierele executabile (cele cu extensia *.exe*) ca programe CGI. Pentru aceasta vom edita fişierul *httpd.conf* (localizat în directorul *conf* al serverului), inserând în secţiunea `<IfModule mod_cgi.c>...</IfModule>` liniile de mai jos:

```
AddHandler cgi-script .exe
AddType application/x-httpd-cgi .exe
```

Putem testa funcţionalitatea serviciului folosind adresa *http://localhost/SalutWS.exe/wsdl/ISalut* într-un browser Web. Va trebui să obţinem fişierul WSDL generat automat de program (avizăm cititorul că formatul WSDL nu respectă standardele în vigoare, ci o versiune mai veche a specificaţiei, actualmente considerată demodată).

Un client Delphi pentru serviciul implementat

În continuare, vom detalia maniera de exploatare a serviciului printr-un client Delphi. Vom crea un *form* pe care vom plasa un buton de execuţie, la apăsarea căruia să se invoce metoda *NumaraZile* a serviciului de mai sus. De asemenea, din *tab*-ul *WebServices* al barei de componente (*Component Palette*), vom alege componenta *HTTPRIO*, responsabilă cu transferurile de date aflate la distanţă via HTTP. În fereastra *Object Inspector*, va trebui să editäm valorile următoarelor proprietăţi asociate componentei amintite (a se urmări cele din figura alăturată):



– *WSDLLocation* – reprezintă locaţia fişierului WSDL asociat serviciului (putem introduce URL-ul *http://localhost/SalutWS.exe/wsdl/ISalut*);
– *Service* – desemnează numele serviciului (mediul generează o listă de opţiuni pe

baza locației WSDL de mai sus, astfel încât vom selecta pentru situația de față unica intrare *ISalutservice*);
– *Port* – reprezintă punctul terminal al serviciului (ca și mai sus, din lista generată, alegem *ISalutPort*).

Pentru a putea utiliza efectiv serviciul Web via *HTTPRIO*, va trebui să recurgem la un alt asistent, denumit *Web Services Importer* (vezi figura 24), responsabil cu generarea unui *unit* de accesare a metodelor puse la dispoziție de serviciu. Asistentul va solicita adresa Web ori numele de fișier local al documentului WSDL corespunzător serviciului importat. La apăsarea butonului *Generate* va fi creat un *unit*, pe care-l denumim *SalutImport.pas*. Codul-sursă al acestuia este următorul:

```
unit SalutImport;

interface
uses Types, XSBuiltIns;
type
  { interfața invocabilă }
  ISalut = interface(IInvokable)
    ['{AE775C10-3E55-41C4-8494-2A57E658F959}']
    function Salut: WideString;  stdcall;
    function NumaraZile: Byte;  stdcall;
  end;

implementation
uses InvokeRegistry;
initialization
  { înregistrarea interfeței }
  InvRegistry.RegisterInterface(TypeInfo(ISalut),
    'urn:salut-ISalut', '');
end.
```

Mai urmează să scriem liniile care realizează efectiv invocarea, folosindu-ne de interfața *ISalut*:

```
unit SalutClient;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, Rio, SoapHTTPClient, StdCtrls;
type
  { fereastra de dialog }
  TFormClient = class(TForm)
```

```
    buton: TButton;
    HTTPRIO: THTTPRIO;
    procedure butonClick(Sender: TObject);
  end;
var
  FormClient: TFormClient;
implementation
{$R *.dfm}
uses SalutImport;

{ procedura de tratare a apăsării butonului }
procedure TFormClient.butonClick(Sender: TObject);
var Salut: ISalut;
begin
  try
    Salut := (HTTPRIO as ISalut);
    { afişăm rezultatul primit de la metoda
      serviciului Web invocat }
    MessageDlg('Au mai rămas ' +
      IntToStr(Salut.NumaraZile) + ' de zile.',
      mtInformation, [mbOk], 0);
   except
      on E: Exception do { tratăm erorile }
        MessageDlg('A apărut o eroare:' + E.Message,
          mtError, [mbAbort], 0);
   end;
end;
end.
```

După compilare, vom putea testa programul aşa cum se poate remarca din captura-ecran de mai jos:



Figura 25. *Interfaţa-utilizator a clientului de invocare a serviciului Web*

Invocarea din PHP a serviciului implementat în Delphi

La finalul acestei secțiuni vom concepe un program PHP care va recurge la extensia *Soap* pentru apelarea metodelor serviciului de mai sus. Ca și în exemplele anterioare, codul-sursă este simplu:

```
$client = new SoapClient(null,
     array('location' =>
       'http://localhost/SalutWS.exe/soap/ISalut',
       'uri' => 'urn:salut-ISalut'));
   $rez = $client->__call('Salut', array());
   echo "<p>Am primit mesajul:
     <strong>$rez</strong></p>";
   $zile = $client->__call('NumaraZile', array());
   echo "<p>Au mai rămas $zile de zile până la
     finalul anului.</p>";
```

După cum se remarcă, serviciul Web este disponibil la adresa *http://localhost/SalutWS.exe/soap/ISalut*, spațiul de nume fiind automat generat de mediul Delphi.

Un posibil rezultat poate fi urmărit în figura 26.


Un client SOAP - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/php-delphi/salut-client.php
Disable Cookies CSS Forms Images Information Miscellane
http://localhost/S...WS.exe/wsdl/ISalut
Un client SOAP
Am primit mesajul: Salut din Delphi, prin SOAP!
Au mai rămas 122 de zile până la finalul anului.
Done
0 errors / 0 warnings

Figura 26. *Rezultatul obținut după execuția metodelor serviciului Web*

3. Invocarea serviciilor Web în contextul AJAX

3.1. *Punerea problemei*

Având la dispoziție serviciul Web privitor la operații cu fișiere (prezentat anterior în cadrul acestui capitol), am dori să verificăm în mod asincron existența unui fișier, pe baza numelui furnizat de utilizator via un formular electronic. Grație AJAX (*Asynchronous JavaScript And XML*), aceasta se poate realiza fără a reîncărca întreaga pagină Web. Pe baza unor date preluate de pe servere aflate la distanță, se poate reactualiza în manieră dinamică arborele DOM (*Document Object Model*) asociat documentului XHTML redat de programul de navigare.

După o succintă prezentare a suitei de tehnologii AJAX, vom furniza detaliile de implementare a unui *script* PHP care va reprezenta puntea de legătură între documentul Web încărcat în browser și serviciul Web scris în Perl.

3.2. *Caracteristici importante ale suitei de tehnologii AJAX*

Componente

Termenul AJAX desemnează o suită de tehnologii deschise, încorporând următoarele:

– diverse limbaje standardizate de prezentare a datelor, precum XHTML și CSS (*Cascading Style Sheets*);
– redarea și interacțiunea via standardul DOM;
– interschimbul și manipularea de date prin XML și/sau XSLT (*Extensible Stylesheet Language Transformations*);
– transferul asincron de date via obiectul *XMLHttpRequest*;
– procesarea prin limbajul ECMAScript/JavaScript.

Dacă aplicațiile Web convenționale erau construite având ca punct central serverul, navigarea fiind bazată pe încărcarea de pagini Web (cu conținuturi statice sau generate dinamic) solicitate serverului, în cazul AJAX aplicația rezidă chiar în cadrul paginii, procesarea având loc la nivelul clientului, iar o parte din date sunt stocate tot de către acesta. Astfel, avantajele majore aduse de AJAX sunt cele legate de eliminarea reîncărcării paginilor, de facilitarea interacțiunii cu utilizatorul, de micșorarea timpilor morți etc. Din punctul de vedere AJAX, întreaga aplicație se regăsește la nivelul browser-ului, ceea ce poate avea repercursiuni nedorite privind sincronizarea serverului cu clientul. De cele mai

multe ori este practic imposibil să se cunoască starea (comportamentul) programelor de pe server. Alte dezavantaje vizează dependența de tehnologia JavaScript (implementată în mod diferit de fiecare browser în parte), dificultatea testării și depanării aplicațiilor, necesitatea folosirii navigatoarelor de ultimă generație, accesul oricui la codul-sursă (ceea ce poate cauza probleme de securitate) și imposibilitatea, în unele situații, de a realiza *bookmark*-uri ori de a folosi mecanismul de istoric al navigării (*history*).

Succesul suitei de tehnologii AJAX provine din promovarea sa de către aplicații Web populare precum *Amazon*, *Basecamp*, *Flickr*, *Gmail*, *Google Maps*, *Google Suggest*, *Netflix*, *Rollyo*, *Writely*, *Yahoo!* și *Zimbra*.

Obiectul XMLHttpRequest

Componenta de bază este obiectul *XMLHttpRequest* pus la dispoziție practic de către orice *browser* Web actual (a apărut în premieră ca obiect ActiveX inclus în Internet Explorer 5, iar apoi a fost adoptat și de Mozilla 1.0 și Safari 1.2). Acest obiect permite realizarea de cereri HTTP (*e.g.*, GET și POST) dintr-un program rulând la nivel de client (program de navigare) spre o aplicație de pe server, într-un mod asincron.

În mod uzual, datele vehiculate între programele client și server sunt marcate în XML, dar pot fi adoptate și alte abordări: reprezentări HTML, JSON (*JavaScript Object Notation*), CSV (*Comma Separated Values*) ori diverse metode de serializare. Atunci când se utilizează HTML ca mijloc de reprezentare a datelor, AJAX se mai numește și AHAH (*Aynchronous HTML and HTTP*). JSON reprezintă un subset al limbajului JavaScript (în fapt, al versiunii standardizate, cunoscută sub numele de ECMAScript), avantajul fiind că datele reprezentate în JSON sunt simple șiruri de caractere ce pot fi convertite ușor în (tablouri de) obiecte JavaScript. Astfel dispare necesitatea procesării conținutului XML obținut de la server.

Principalele metode care vor fi folosite efectiv în contextul AJAX sunt următoarele:

– *open()* inițiază (deschide) o conexiune HTTP cu serverul, trimițând o cerere (uzual, GET sau POST) către acesta; aplicația de pe server cu care va avea loc schimbul de date va fi desemnată de un URI, după cum vom vedea mai jos;
– *send()* transmite date, în manieră asincronă, către aplicația rulând pe server;
– *abort()* abandonează transferul curent;
– *setRequestHeader()* specifică diverse câmpuri ale antetului HTTP; de exemplu, se poate stabili o anumită valoare particulară pentru câmpul *Content-Type*, aspect util atunci când dorim să manipulăm alte tipuri de conținuturi, de exemplu JSON ori CSV.

Proprietățile de bază ale obiectului sunt:

– *readyState* conține codul de stare a transferului (*e.g.*, valoarea 4 specifică faptul că transferul datelor s-a efectuat complet) – a se consulta şi tabelul de mai jos;

Valoarea *readyState*	**Stare**	**Descriere**
0	UNINITIATED	Obiectul XML a fost creat, dar procesul de încărcare nu a fost încă pornit.
1	LOADING	Încărcarea este în progres, dar analiza XML nu a început.
2	LOADED	Încărcarea s-a terminat, însă arborele DOM nu a fost creat.
3	INTERACTIVE	Arborele DOM a fost generat, dar numai o parte dintre elemente au fost analizate.
4	COMPLETED	Documentul XML a fost complet încărcat şi analizat.

– *status* reprezintă codul de stare HTTP întors de serverul Web; de exemplu, 200 (*Ok*), 404 (*Not Found*), 500 (*Server Internal Error*);
– *statusText* desemnează şirul de caractere ce explică în limbaj natural codul de stare obținut;
– *onreadystatechange* specifică funcția ce va fi invocată la modificările privind schimbarea stării de transfer a datelor (datele sunt transmise în manieră asincronă şi deci programul de la nivel de client trebuie să poată fi notificat asupra schimbării evenimentelor care pot surveni pe parcursul transferului).

Utilizări

Caracteristica importantă oferită de AJAX este cea de a schimba doar anumite părți ale paginii Web.

Una din tehnicile AJAX populare este şi cea a indicării vizuale a faptului că un fragment dintr-un document a fost modificat. Astfel, pe baza unei nuanțe de culoare, care dispare în timp, utilizatorului i se poate semnala că un paragraf dintr-un text are conținutul schimbat (actualizat). Uzual, se foloseşte galbenul; de aceea, acest şablon de proiectare se mai numeşte şi YFT (*Yellow Fade Technique*), generalizarea sa întitulându-se FAT (*Fade Anything Technique*).

O altă utilizare este cea a reîmprospătării automate (eventual, periodice) a unei zone dintr-o pagină. De exemplu, atunci când a apărut o nouă ştire recepționată via un emițător Atom/RSS (*Really Simple Syndication*) ori s-a modificat

o informație de interes pentru utilizator (curs valutar, temperatură, scor la un meci, voturi acordate unei piese muzicale etc.).

Tot prin intermediul AJAX, putem realiza un câmp interactiv cu auto-completarea datelor (*auto-completion*), în funcție de cele introduse deja de către utilizator. Astfel se emulează bine-cunoscutul control *combo-box*, oferindu-se o listă de sugestii în funcție de caracterele tastate. Situl *Google Suggest* folosește intensiv această tehnică. În acest context, putem menționa că AJAX ne poate sluji și pentru validarea diverselor informații obținute de la utilizator.

Altă facilitate ușor de implementat via AJAX este cea de a oferi posibilitatea efectuării de operații *drag-and-drop* asupra unor zone prestabilite ale paginii Web. În conjuncție cu o bibliotecă JavaScript precum *DOM-Drag*, se poate astfel implementa – la nivel de aplicație Web – un comportament similar unei interacțiuni convenționale de tip MDI (*Multiple Document Interface*).

Diverse alte întrebuințări ale AJAX sunt dedicate realizării de efecte vizuale (unele dintre ele, sofisticate), similare interacțiunii cu utilizatorul în Mac OS X sau Windows Vista.

Instrumente de dezvoltare

Actualmente sunt disponibile diverse cadre de lucru dedicate dezvoltării aplicațiilor AJAX. La nivelul clientului pot fi folosite diferite biblioteci (*e.g.*, *Dojo*, *Prototype*, *Rico* ori *Script.aculo.us*), unele dintre ele oferind și suport pentru manipularea facilă a arborilor DOM. Pe partea de server se poate recurge la tehnologii precum CGI, servere de aplicații (de exemplu, programe PHP sau *servlet*-uri) ori *framework*-uri dedicate – vezi mai jos.

Trebuie menționat faptul că o serie de companii deja au început să pună la dispoziție interfețe de programare AJAX specializate, pentru ca dezvoltatorii să integreze facil conținuturi provenite de pe diverse situri în cadrul paginilor proprii. Un exemplu notabil este *Google AJAX Search API*, prin care se poate avea acces la serviciul Web de căutare direct la nivel de client, în stilul AJAX.

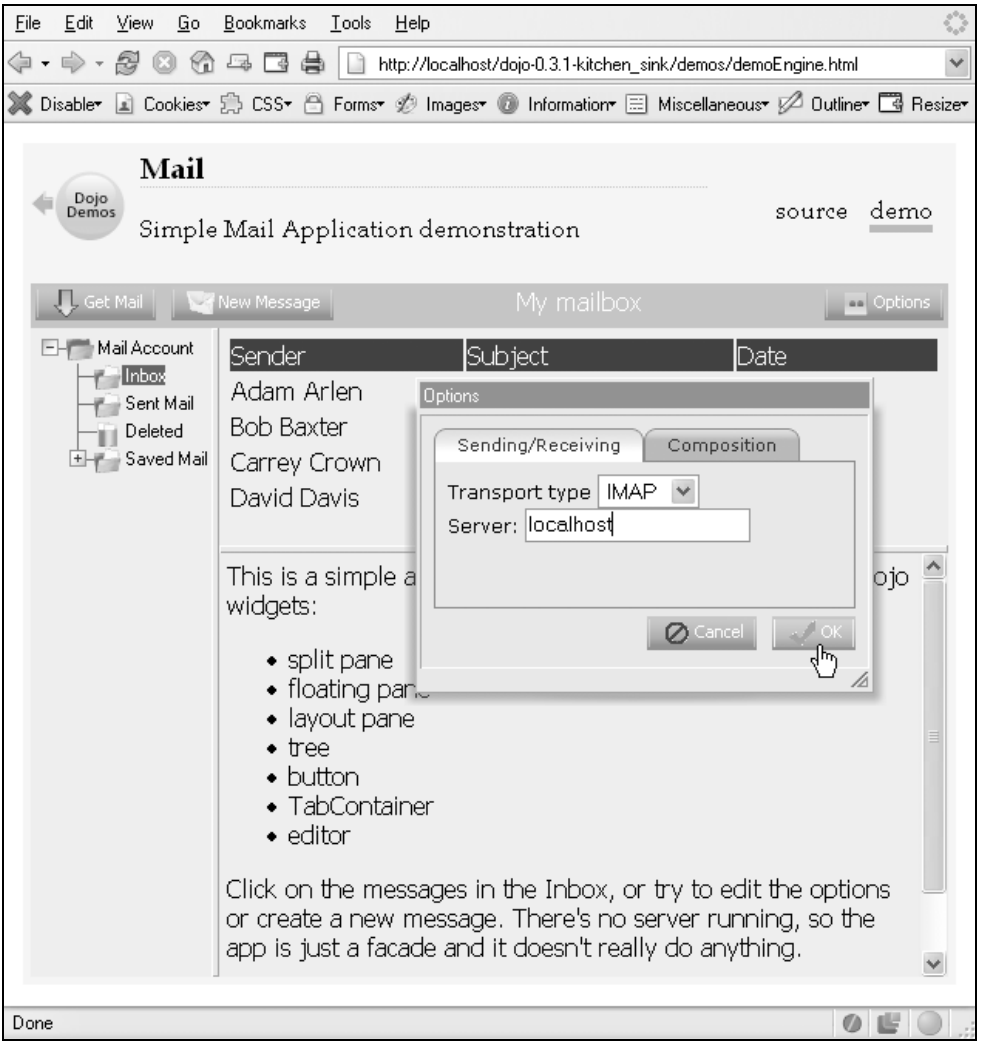


Figura 27. *O aplicație Web demonstrativă construită cu biblioteca* Dojo *folosind diverse tehnici AJAX*

3.3. *Invocarea asincronă a unui serviciu Web via AJAX*

Folosirea efectivă a obiectului XMLHttpRequest *și a unui intermediar scris în PHP*

Vom considera scenariul: un utilizator dorește să verifice existența unui fișier aflat pe un server, pentru aceasta trebuind să completeze un formular care îi va solicita numele fișierului în cauză. Prin intermediul AJAX, vom apela un program PHP ce va reprezenta un client pentru serviciul Web privitor la operații cu fișiere, detaliat în subcapitolul 1.

La apariția evenimentului de pierdere a *focus*-ului asupra câmpului de introducere a numelui de fișier, prin intermediul unui program JavaScript se va realiza un transfer asincron către programul PHP rulând pe server, aplicație care va întoarce răspunsul privind existența numelui de fișier. În acest caz, utilizatorul va primi imediat *feedback* pozitiv, suplimentar având posibilitatea redenumirii fișierului respectiv.

Formularul Web de interacțiune cu utilizatorul are următoarea structură:

```
<form action="redenumire.php" method="get">
   Numele fişierului : <input type="text" name="nume"
      title="Introduceţi un nume de fişier"
      onblur=
        "javascript:verificaExist (this.value, '')" />
   <span class="ascuns" id="fisierExistent">
      Fişierul există.
      <input type="checkbox" value="da"
        name="redenumire" checked="checked"
        title="Bifaţi dacă doriţi redenumirea
          fişierului" />Redenumim?
   </span>
   <br /><input type="submit" value="Execută" />
</form>
```

În cadrul unui element `<span>` vom insera construcțiile XHTML ce vor conține mesajul afișat utilizatorului în cazul în care fișierul există, putându-se bifa opțiunea de redenumire a acelui fișier. Inițial, acest element va fi ascuns și va fi afișat prin modificarea proprietății de stil *display* oferită de CSS.

Clasele de proprietăți CSS sunt specificate în antetul documentului XHTML în maniera de mai jos:

```
input[type="text"]:focus {
  background:   #EEE;
  border-color: #000;
}
.exista { /* pentru semnalarea exist. fişierului */
  display:      block;
  background:   #CCF;
  border-width: 1px;
  border-style: dotted;
  padding:      0.2em;
  width:        233px;
}
.ascuns { /* pentru ascunderea mesajului */
  display:      none;
}
```

La apariţia evenimentului *onblur* va fi apelată funcţia JavaScript *verificaExist()* care va avea ca responsabilitate efectuarea transferului asincron către un program PHP rulând pe server ce va apela metoda de verificare a existenţei fişierului, metodă implementată de serviciul Web conceput în limbajul Perl. Atât cererea, cât şi rezultatul obţinut vor fi marcate în XML.

Codul-sursă al funcţiei *verificaExist()* este următorul:

```
function verificaExist (nume, raspuns) {
  // avem un răspuns?
  if (raspuns != '') {
    // preluăm via metodele DOM
    // elementul cu id='fisierExistent'
    // pentru a afişa mesajul dorit
    mesaj = document.getElementById ('fisierExistent');
    // schimbăm stilul de afişare (în caz de succes
    // vor fi aplicate proprietăţile CSS din
    // clasa 'exista', altfel textul va fi ascuns)
    mesaj.className =
      (raspuns == 1) ? 'exista' : 'ascuns';
  }
  else {
    // nu e nici un răspuns setat, vom verifica
    // trimiţând o cerere asincronă către server
    incarcaXML ('/verifica_exist.php?nume=' + nume);
  }
}
```

Cererea asincronă către programul PHP va fi realizată de funcţia *incarcaXML()* având codul dat în continuare:

```
var cerere; // cererea către serverul Web
// încarcă un document XML desemnat de 'url'
function incarcaXML (url) {
  // verificăm existenţa obiectului XMLHttpRequest
  if (window.XMLHttpRequest) {
     // există suport nativ (Mozilla Firefox)
     cerere = new XMLHttpRequest ();
     // se stabileşte funcţia de tratare
     // a stării încărcării
     cerere.onreadystatechange = trateazaCererea;
     // preluăm documentul prin metoda GET
     cerere.open ("GET", url, true);
     cerere.send (null);
  }
  else if (window.ActiveXObject) {
     cerere = new ActiveXObject ("Microsoft.XMLHTTP");
     // se poate folosi obiectul ActiveX din MSIE
```

```
      if (cerere) {
         // preluăm documentul prin metoda GET
         cerere.onreadystatechange = trateazaCererea;
         cerere.open ("GET", url, true);
         cerere.send ();
      }
   }
}
```

Se verifică disponibilitatea obiectului *XMLHttpRequest* în cadrul navigatorului Web, iar maniera de transfer se va realiza prin metoda GET a protocolului HTTP. Tratarea evenimentelor de transmisie asincronă a datelor are loc în funcția *trateazaCererea()* a cărei implementare este furnizată mai jos:

```
function trateazaCererea () {
  // verificăm dacă încărcarea s-a terminat cu succes
  if (cerere.readyState == 4) {
    // verificăm dacă am obținut codul de stare '200 Ok'
    if (cerere.status == 200) {
    // procesăm datele recepționate (preluăm via DOM
    // elementul-rădăcină al documumentului XML)
   raspuns = cerere.responseXML.documentElement;
   metoda = raspuns.getElementsByTagName(
         'metoda')[0].firstChild.data;
   rezultat = raspuns.getElementsByTagName(
         'rezultat')[0].firstChild.data;
   // evaluăm metoda
   eval ( metoda + '(\'\', rezultat)');
    }
    // se pot trata şi alte coduri (404, 500 etc.)
    else {
   alert ("A apărut o problemă la transferul XML:\n"
         + cerere.statusText);
    }
  }
}
```

Datele XML care vor fi recepționate reprezintă un nume de metodă – în acest caz, *verificaExist()* – și rezultatul întors de aceasta. Aceste date vor fi prelucrate doar în caz de succes (codul de stare HTTP va fi 200).

Programul PHP ce verifică existența fișierului pe baza numelui recepționat de la utilizator via AJAX are următorul cod-sursă:

```
<?php
require_once('lib/nusoap.php');
// trimitem tipul conținutului
```

```
header ('Content-type: text/xml');

// funcţie care verifică dacă există un fişier
function verifica ($nume) {
  // instanţiem un client SOAP
  $client = new soapclient(
     'http://endirra.ro:3333/xml-files.pl');
  // preluăm posibilele erori
  $eroare = $client->getError();
  if ($eroare)
    return 0;
  // stabilim parametrii de intrare ai metodei invocate
  $param = array('filename' => $nume);
  // stabilim spaţiul de nume folosit
  $spatiu_nume = 'http://127.0.0.1:3333/XMLFiles/';
  // realizăm apelul
  $rezult = $client->call('CheckIfExists',
    array('parameters' => $param),
    $spatiu_nume, '', true);
  // verificăm dacă există vreun fault
  if ($client->fault)
    return 0;
  // eroare la executie?
  $eroare = $client->getError();
  if ($eroare)
    return 0;
  // întoarcem rezultatul oferit de metoda invocată
  return $rezult;
}
// generăm conţinutul XML
echo '<?xml version="1.0" ?>';
?>
<raspuns>
  <metoda>verificaExist</metoda>
  <rezultat>
    <?php echo verifica ($_REQUEST['nume']); ?>
  </rezultat>
</raspuns>
```

Trebuie invocată metoda *CheckIfExists* implementată de serviciul Web creat în Perl.

Programul PHP recurgând la funcționalitățile oferite de biblioteca *NuSOAP* devine în acest moment client al serviciului, funcționând ca un intermediar între *script*-ul JavaScript rulând în browser și serviciul Web executat pe un server aflat la distanță. Cele trei categorii principale de entități de calcul implicate în această acțiune sunt ilustrate de figura 28.

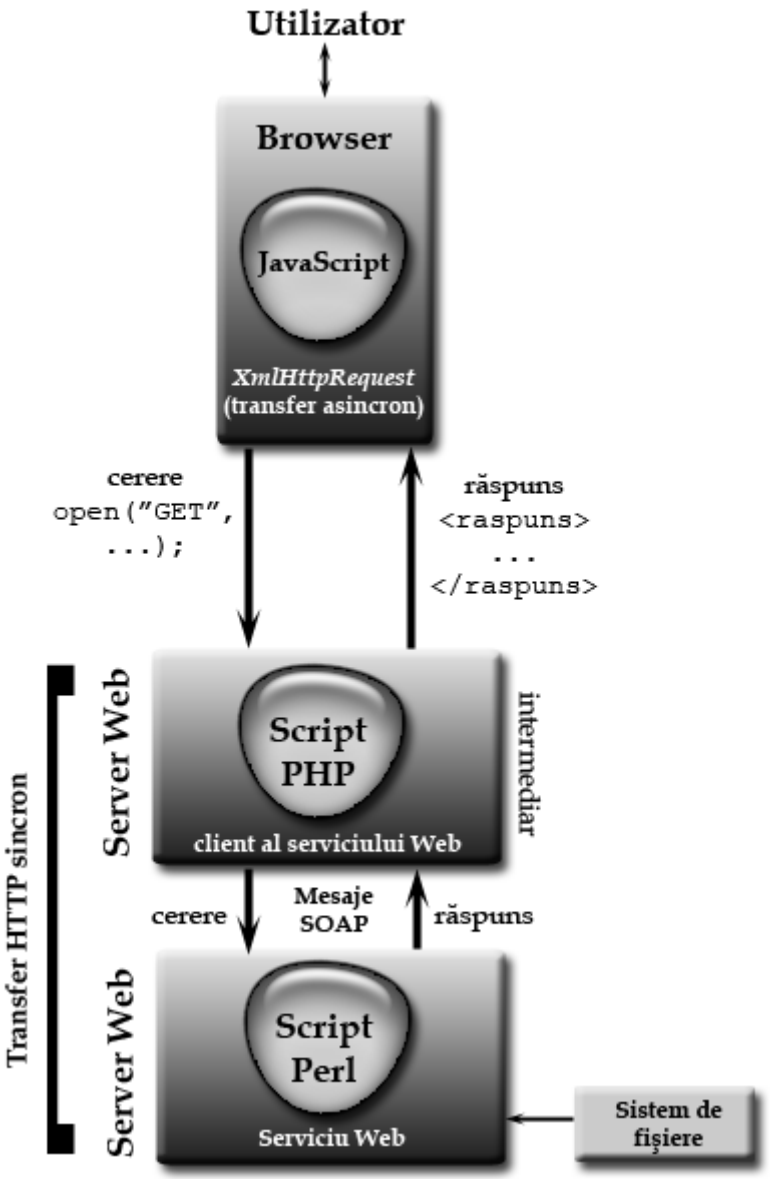


Figura 28. *Relațiile între programul JavaScript,* script-*ul PHP și serviciul Web implementat în Perl*

Tipul conținutului generat va fi XML, așteptat pe partea de client de către programul JavaScript.

Figura următoare reprezintă o captură-ecran oferind mesajul obținut de utilizator în cazul în care introduce un nume de fișier existent, cu posibilitatea redenumirii acestuia.

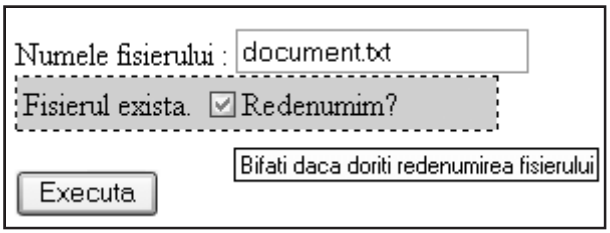


Figura 29. *Mesajul obținut în urma introducerii unui nume de fișier existent*

Apelând *script*-ul PHP în mod explicit – introducând în browser o adresă de genul *http://endirra.ro/verifica_exist.php?filename=inexistent* –, obținem documentul XML ilustrat de figura 30. Acest document încapsulează de fapt răspunsul prelucrat în cadrul programul JavaScript descris anterior.

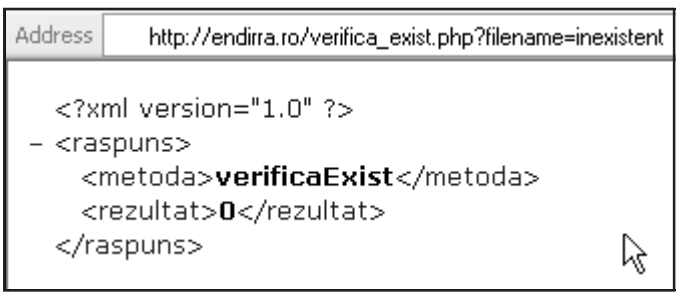


Figura 30. *Documentul XML furnizat de programul PHP de verificare a existenței unui fișier*

Din cele descrise mai sus, se poate observa faptul că *script*-ul PHP funcționează ca un intermediar (*proxy*), realizând inter-operabilitatea între programul JavaScript rulat în cadrul browser-ului și serviciul Web Perl invocat pe un server diferit de cel în care se execută codul PHP. Am ales modelul descris aici, deoarece serviciul Web poate să nu fie disponibil pe aceeași mașină pe care rulează codul PHP invocat prin obiectul *XMLHttpRequest*.

La apăsarea butonului de tip *submit* va fi apelat *script*-ul *redenumire.php*, care va trebui să invoce metoda de redenumire a fișierului, metodă implementată de serviciul Web. Scrierea acestui program rămâne ca temă de acasă pentru cititorul interesat.

3.4. *Utilizarea bibliotecii* Prototype

O soluție alternativă la precedenta este aceea în care recurgem la biblioteca *Prototype*, care pune la dispoziție o suită de clase și extensii DOM (*Document Object Model*) pentru facilitarea unei interacțiuni Web flexibile pe partea client, în

JavaScript. Una dintre funcționalitățile pe care o vom utiliza se referă tocmai la implementarea unui mecanism robust, independent de navigator, de transfer asincron de date în stilul AJAX. Astfel, pagina Web care solicită utilizatorului un nume de fișier pentru a verifica existența sa pe server are următorul cod-sursă:

```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="ro" xml:lang="ro">
<head>
<title>AJAX + SOAP</title>
<style type="text/css">
.exista { /* pt. semnalarea existenței fișierului */
  display:      block;
  background:   #CCF;
  border-width: 1px;
  border-style: dotted;
  padding:      0.2em;
  width:        233px;
}
.ascuns { /* pt. ascunderea mesajului */
  display: none;
}
</style>
<!-- folosim biblioteca Protoype 1.4.0 -->
<script type="text/javascript"
        src="prototype.js"></script>
<script type="text/javascript">
// lista de tratare a evenimentelor
// de încărcare asincronă
var globalHandlers = {
      // la crearea conexiunii cu serverul
  onCreate: function () {
       // afișăm un mesaj temporar
       Element.show ('asteptare');
  },
      // la apariția unei erori
  onFailure: function() {
       alert ('Eroare de transmitere.');
  },
      // la apariția unei excepții
  onException: function() {
            alert ('A apărut o excepție.');
  }
};
```

```
// funcţie care verifică existenţa
// numelui de fişier via AJAX
function verificaExist () {
  // înregistrăm funcţiile de tratare a evenimentelor
  Ajax.Responders.register (globalHandlers);
  // instanţiem obiectul AJAX
  var obAjax = new Ajax.Request (
    '/ajax/verifica_exist.php', // URI-ul invocat
    {
   method: 'get',
   // parametrii transmişi spre server
   parameters: 'nume=' + $F('nume'),
   // metoda apelată la terminarea transferului
   onComplete: afiseaza
    });
}

// funcţie responsabilă cu afişarea rezultatului
function afiseaza(rasp) {
  // ascundem mesajul temporar
  Element.hide ('asteptare');
  // preluăm valoarea rezultatului
  // printr-o expresie regulată,
  // deoarece răspunsul e un şir de caractere
  var pattern =
        new RegExp("\<rezultat\>1\<\/rezultat\>");
  // modificăm clasa CSS corespunzătoare
  // elementului <span> iniţial ascuns
  $('fisierExistent').className =
     (pattern.test(rasp.responseText)) ?
        'exista' : 'ascuns';
}
</script>
</head>
<body>
<form action="redenumire.php" method="get">
   Numele fişierului : <input type="text" id="nume"
      title="Introduceţi un nume de fişier"
      onblur="javascript:verificaExist ()" />
   <span id="asteptare"
  style="display:none;color:#990;font-style:italic">
    Aşteptaţi, vă rugăm...</span>
   <span class="ascuns" id="fisierExistent">
      Fişierul există.
      <input type="checkbox" value="da"
        name="redenumire" checked="checked"
        title="Bifaţi dacă doriţi redenumirea
```

```
                fişierului" />Redenumim?
    </span>
    <br />
    <input type="submit" value="Execută" />
</form>
</body>
</html>
```

Evenimentele privitoare la desfăşurarea transferului de date între client şi server pot fi tratate de funcţii definite de utilizator. În cazul de faţă, la crearea unui conexiuni cu serverul, vom afişa un mesaj temporar ce semnalează utilizatorului faptul că trebuie să aştepte primirea răspunsului (a se vedea figura 31).

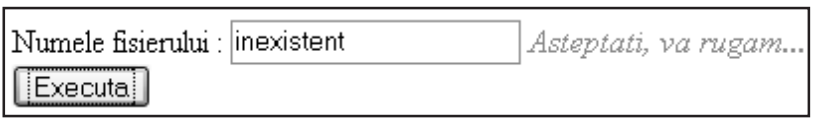


Figura 31. *Semnalarea unui mesaj de aşteptare a realizării unui transfer asincron via AJAX*

Maniera de afişare/ascundere a unui element – `<span id="asteptare">` – este realizată de metodele *show()*, respectiv *hide()* ale clasei *Element* definite de biblioteca *Prototype*. În caz de eroare (*onFailure*) sau de excepţie (*onException*), vom semnala această situaţie utilizatorului via funcţia JavaScript standard *alert()*. Ca şi în exemplul anterior, verificarea existenţei fişierului se realizează în cadrul funcţiei *verificaExist()*. În această situaţie, vom instanţia un obiect *Ajax.Request* ce reprezintă cererea efectuată de program către server, după ce în prealabil am înregistrat funcţiile de tratare a evenimentelor de transfer via *Ajax.Responders.register()*. Argumentele constructorului *Ajax.Request()* sunt, în mod firesc, adresa *script*-ului PHP ce va fi invocat pe server (acelaşi program ca mai sus), metoda HTTP folosită, parametrii de intrare (în cazul nostru, numele fişierului) şi funcţia ce va fi apelată la finalizarea transferului asincron. Valoarea numelui de fişier este preluată din formular prin intermediul construcţiei *$F()* specificate de *Prototype*. În acest mod, putem accesa comod câmpuri din formularele Web.

După obţinerea rezultatului va fi executată funcţia *afiseaza()*, care are drept argument un obiect conţinând răspunsul primit din partea serverului, în cazul nostru, un document XML. Proprietatea *responseText* reprezintă şirul de caractere recepţionat din partea *script*-ului PHP, adică documentul XML ilustrat în figura 30. Printr-o expresie regulată simplă, preluăm valoarea elementului `<rezultat>`, pentru a schimba clasa CSS corespunzătoare marcatorului `<span>` responsabil cu semnalarea existenţei fişierului şi inserării butonului de bifare a redenumirii

acestuia. Construcția *$()* este o scurtătură pentru metoda *getElementById()* specificată de recomandarea DOM, fiind folosită pentru a selecta un anumit element din cadrul documentului XHTML; a se revedea și capitolul 2.

O captură-ecran a dialogului între client și server este pusă la dispoziție de figura 32, în care se poate remarca și interfața extensiei *Firebug* pentru navigatorul Firefox, folosită în situația de față pentru a lista apelurile AJAX realizate și posibilele erori survenite. Această extensie poate fi utilizată cu succes pentru depanarea oricăror programe JavaScript.

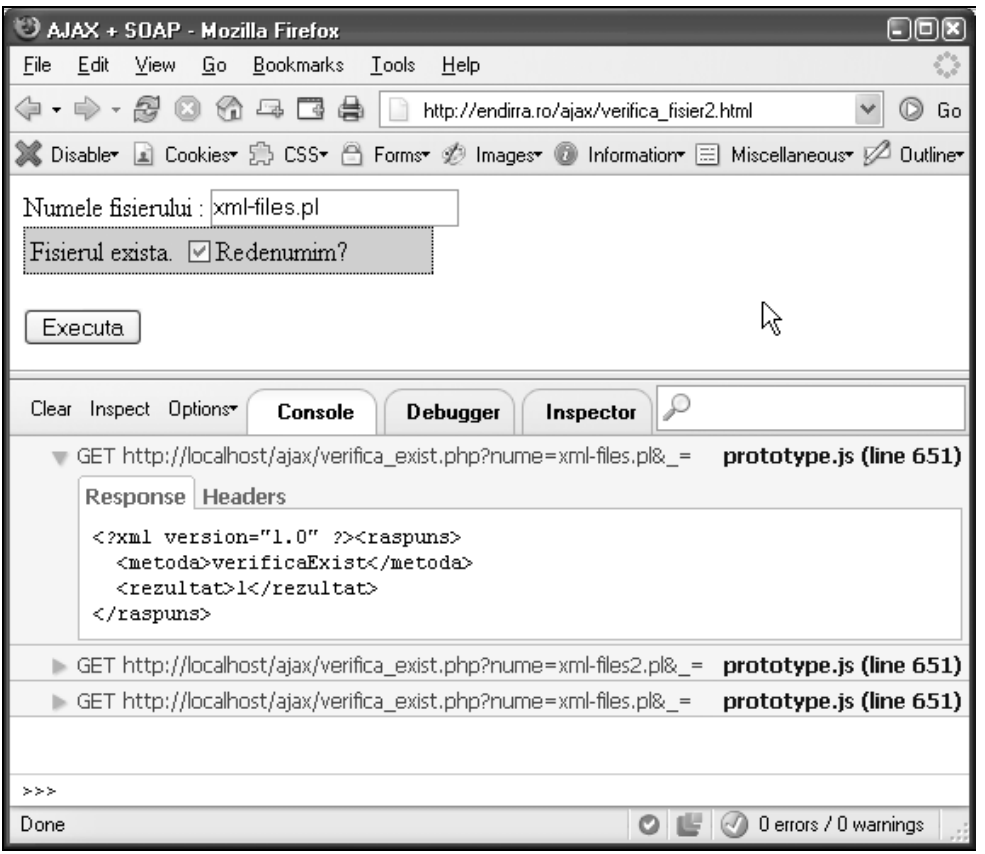


Figura 32. *Trasarea apelurilor AJAX via extensia Firebug pentru navigatorul Firefox*

3.5. *Invocarea directă, din JavaScript, a unui serviciu Web*

În continuare, ne propunem să invocăm direct un serviciu Web via metoda POST. Singura restricție este ca URI-ul folosit de obiectul *XMLHttpRequest* pentru trimiterea mesajului SOAP de cerere să aibă același domeniu cu cel al

serviciului (cu alte cuvinte, serviciul Web nu poate fi localizat decât pe mașina pe care rezidă și programul JavaScript ce recurge la AJAX).

Ca exemplificare, presupunem că dorim să aflăm stocul unui sortiment de portocale, recurgând la metoda *furnizeazaSortiment()* implementată de serviciul Web prezentat în cadrul secțiunii 1.2. Programul JavaScript folosit la invocarea serviciului Web are codul-sursă de mai jos (se poate observa că structura generală a codului e similară celei prezentate mai sus, cu deosebirea că în acest caz vom iniția o cerere POST spre server, adresa indicată fiind cea a serviciului Web):

```
// adresa de apel al serviciului Web
const URISERV =
  'http://localhost/sxml/oranges-server.php';
// adresa de bază a serviciului Web
const URIBSRV = 'http://localhost/sxml/';
// spațiul de nume al serviciului Web
const URISPN = 'urn:portocale.info';

// generăm mesajul SOAP de cerere
// care va fi trimis spre server
function genMesaj (numeDoc) {
// construim 'de mână' mesajul SOAP de cerere
var mesaj = "<soap:Envelope xmlns:xsi=\""
  + "http://www.w3.org/2001/XMLSchema-instance\" "
  + "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" "
  + "xmlns:soap=
    \"http://schemas.xmlsoap.org/soap/envelope/\">\n"
  + "<soap:Body>\n"
  + "<furnizeazaCantit " + "xmlns=\"" +
    URISPN + "\">\n"
  + "<sortiment>" + numeDoc + "</sortiment>\n"
  + "</furnizeazaCantit>\n"
  + "</soap:Body>\n"
  + "</soap:Envelope>\n";
return mesaj;
}

// invocăm metoda de validare
function invoca (numeDoc) {
var mesaj = genMesaj (numeDoc);
try {
  if (window.XMLHttpRequest) { // doar pentru Firefox
     cerere = new XMLHttpRequest ();
  }
  // stabilește funcția de tratare a stării încărcării
  cerere.onreadystatechange = trateazaCererea;
```

```
  // trimitem cererea SOAP prin metoda POST
  cerere.open ("POST", URISERV, true);
  // stabilim tipul conţinutului
  cerere.setRequestHeader("Content-Type", "text/xml");
  // stabilim valoarea câmpului SOAPAction
  cerere.setRequestHeader("SOAPAction",
    URIBSRV + 'furnizeazaCantit');
  cerere.send (mesaj);
  document.getElementById("txtCerere").value = mesaj;
}
catch (except) {
  alert ('A intervenit o excepţie: ' + except);
}
}

// funcţie care tratează starea conexiunii cu serverul
function trateazaCererea () {
// transferul s-a efectuat complet
if (cerere.readyState == 4) {
  var raspuns = cerere.responseXML;
  // a apărut o eroare?
  if (cerere.status != 200) {
    alert("A apărut o eroare...\n" +
       cerere.statusText +
       cerere.getAllResponseHeaders());
    return;
  }
  // furnizăm răspunsul primit de la server
  document.getElementById("txtRaspuns").value =
    raspuns.getElementsByTagName(
      'Body')[0].firstChild.firstChild.firstChild.data;
    return;
  }
}
```

Mesajul ce va fi transmis către server este „confecţionat” manual, construind în acest mod plicul SOAP. În caz de eroare la întoarcere răspunsului, afişăm şi anteturile HTTP primite de la server via *getAllResponseHeaders()*. Valoarea stocului de portocale reprezintă în fapt elementul strănepot al elementului *Body* al răspunsului SOAP (folosim proprietatea *firstChild* pusă la dispoziţie de modelul DOM).

Formularul XHTML care preia de la utilizator numele unui sortiment de portocale şi afişează răspunsul primit, inclusiv structura cererii SOAP efectuate, are următorul cod:

```
<form>
   Numele sortimentului :
     <input type="text" name="nume"
```

```
        title="Introduceti un sortiment de portocale"
        onblur="javascript:invoca (this.value)" />
    <br />
    <!-- conţine datele privitoare la cererea SOAP -->
    <textarea rows="10" cols="60"
              id="txtCerere"></textarea>
    <p>Stocul din sortimentul dorit este
     <input type="text" id="txtRaspuns"></input>.</p>
</form>
```

Un posibil rezultat este disponibil în figura de mai jos.

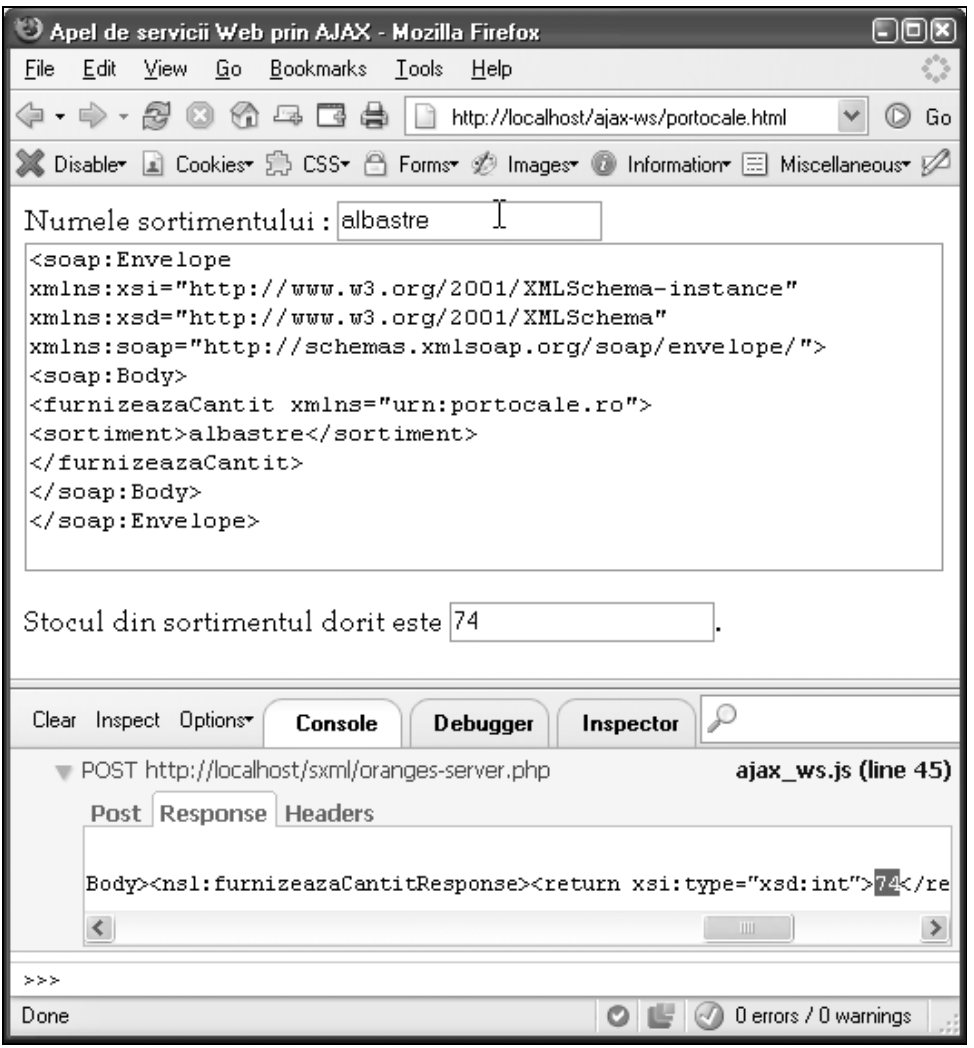


Figura 33. *Rezultatul primit de la o metodă a unui serviciu Web invocat direct via* XMLHttpRequest

3.6. *Transferuri asincrone prin* Atlas ASP.NET

La finalul acestui subcapitol vom prezenta succint cadrul de lucru *Atlas ASP.NET*, conceput special de Microsoft pentru realizarea de aplicații AJAX în contextul ASP.NET versiunea 2.0, dar putând fi folosit și în conjuncție cu alte tehnologii de programare a aplicațiilor Web, de exemplu PHP – a se vizita adresa *http://www.shankun.com/Atlas_Php_2.aspx*. De asemenea, partea JavaScript facilitând transferurile asincrone e compatibilă cu mai multe navigatoare, fără a trebui să recurgem în mod necesar la Internet Explorer.

Pentru a utiliza Atlas, cunoștințele pe care trebuie să le dețină programatorul în ceea ce privește tehnologia ASP.NET vor putea fi minimale.

Pentru redarea datelor la nivelul clientului, se oferă un set de controale de interfață, în stilul ASP.NET, ale căror proprietăți pot fi stabilite fie declarativ (prin construcții XML), fie prin intermediul codului JavaScript. Aceste controale vor rula în fapt pe server, mediul având în responsabilitate generarea de marcaje (plus programe JavaScript), în funcție de browser-ul folosit. Funcționalitatea este încapsulată de un *assembly* numit *Microsoft.Web.Atlas.dll*.

O parte dintre controalele deja existente în ASP.NET sunt extinse pentru a oferi facilități suplimentare, în stilul AJAX. De asemenea, câmpurile de tip text pot avea atașat un mecanism de autocompletare a datelor (vezi exemplul de mai jos), iar conținutul unor zone din cadrul documentului Web pot fi reîmprospătate periodic (via *UpdatePanel*). Diverse elemente dintr-o pagină pot interacționa prin intermediul tehnicilor *drag-and-drop*, putând fi astfel mutate/ascunse dinamic. Mai mult decât atât, terțe componente pot îmbogăți paleta controalelor existente; astfel, în Atlas pot fi incluse funcționalități noi.

Accesul la datele de pe server se realizează prin apeluri asincrone de metode ale unor servicii Web locale (codul acestora trebuie să rezide pe aceeași mașină pe care există situl), conform modelului de securitate impus de JavaScript. Aceste servicii Web vor fi construite conform modalităților oferite de ASP.NET. Mediul are rolul de a realiza transferul efectiv al datelor – codificate conform convenției JSON între controalele interactive și server.

Invocarea serviciilor externe se poate realiza prin așa-numita punte (*bridge*), cod inclus în fișiere cu extensia *.asbx*, ce va facilita dialogul cu servicii rulând pe alte calculatoare. Un astfel de exemplu de fapt l-am implementat mai sus, în secțiunea 3.3, dar printr-un intermediar scris în limbajul PHP.

Alte detalii privitoare la arhitectura și modul de programare sunt disponibile în documentațiile și exemplele oferite pe situl Atlas.

Drept exemplificare (o adaptare după una dintre soluțiile din tutorialele oficiale), ne propunem să punem utilizatorului la dispoziție o facilitate de

sugerare a numelor unor sortimente de portocale, pe baza caracterelor deja testate de acesta. E vorba, așadar, de asocierea unui mecanism de autocompletare de date pentru un câmp textual al unui formular Web. Datele privitoare la sortimentele de portocale vor fi furnizate de un serviciu Web, care le va prelua dintr-un fișier text (desigur, scenarii mai complexe vor putea recurge la interogări SQL ori la preluarea informațiilor din surse multiple).

Atlas impune ca serviciul Web utilizat să aibă o unică metodă ce va fi invocată pe server, în manieră asincronă, la apariția unui eveniment (apăsarea unui buton, introducerea unor caractere, la momente periodice de timp, la mișcarea cursorului mouse-ului etc.). Așadar, vom dezvolta un serviciu Web stocat de fișierul *Portocale.asmx*. Pentru aceasta, vom folosi șablonul *„Atlas" Web Site* pentru *Visual Web Developer Express* (funcționează și în Visual Studio .NET), pus la dispoziție în momentul instalării distribuției Atlas preluate de pe Web. Ulterior, vom atașa (*bind*) acest serviciu unui control *autoComplete* dintr-o pagină ASP.NET.

Codul-sursă este următorul – a se observa existența unei singure metode, denumită *furnizeazaSortimente*.

```
<%@ WebService Language="C#"
    Class="Portocale.Portocale" %>
using System;
using System.IO;
using System.Collections;
using System.Web.Services;

namespace Portocale {
  [WebService(Namespace = "urn:portocale.info")]
  [WebServiceBinding(ConformsTo =
    WsiProfiles.BasicProfile1_1)]
  public class Portocale :
    System.Web.Services.WebService {
    // lista sortimentelor încărcate dintr-un fișier
    private static string[] sortimente = null;

    // furnizează o listă de sortimente ale căror nume
    // sunt prefixate de un sir dat
    [WebMethod]
    public String[] furnizeazaSortimente
      (string prefixText, int count) {
      // 'prefixText' și 'count' sunt denumiri
      // standard ai parametrilor de intrare trimiși
      // de controlul 'autoComplete'
      string prefix = prefixText;
      int lungMax = count;
      // încă nu există, încărcăm din fișier
```

```
        if (sortimente == null) {
          String[] temp = File.ReadAllLines(
            Server.MapPath("~/sortim_portocale.txt"));
          // sortăm elementele pentru a putea realiza
          // căutarea binară
          Array.Sort(temp,
            new CaseInsensitiveComparer());
          sortimente = temp;
        }
        // realizăm căutarea şi extragem maximum 'contor'
        // sortimente care se potrivesc prefixului
        // preluat de la client
        int index = Array.BinarySearch(sortimente,
          prefix, new CaseInsensitiveComparer());
        if (index < 0) {
          index = ~index;
        }
        int potriviri;
        for (potriviri = 0; potriviri < lungMax &&
             index + potriviri < sortimente.Length;
             potriviri++) {
          // verificăm dacă începe cu prefixul dat
          if (!sortimente[index +
                potriviri].StartsWith(prefix,
          StringComparison.CurrentCultureIgnoreCase)) {
            break;
          }
        }
        // copiem rezultatul obţinut
        String[] rezultat = new string[potriviri];
        if (potriviri > 0) {
          Array.Copy(sortimente, index, rezultat,
            0, potriviri);
        }
        // transmitem spre client tabloul de şiruri
        return rezultat;
      }
    }
}
```

Desigur, putem invoca în mod explicit metoda definită mai sus. Primul parametru de intrare reprezintă prefixul, care trebuie să se potrivească fiecăruia dintre elementele listei de sortimente, iar al doilea specifică numărul maxim de sortimente ce vor fi întoarse. Datele vor fi sortate, iar căutarea va fi realizată prin metoda căutării binare, indiferent dacă literele sunt minuscule sau majuscule.

Un posibil rezultat, cu *prefixText = "a"* și *count = 10*, este documentul XML dat în continuare:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:portocale.info">
  <string>Acre</string>
  <string>Albastre</string>
  <string>Albe</string>
  <string>Argintii</string>
  <string>Aspre</string>
  <string>Aurii</string>
</ArrayOfString>
```

Va trebui să scriem o pagină ASP.NET care să ofere un formular Web unde utilizatorul să fie îndemnat să furnizeze un nume de sortiment de portocale. La introducerea măcar a unui caracter, pe baza prefixului deja inserat în câmpul formularului, va trebui inițiat un transfer asincron care să invoce metoda *furnizeazaSortimente()*. Rezultatul acesteia va fi afișat ca listă de sugestii într-un * *, utilizatorul putând selecta unul dintre sortimentele apărute.

Codul XHTML al formularului este următorul:

```
<form id="formular" action="" runat="server">
   <div>Introduceți un sortiment:
        <input id="sortiment" type="text" /></div>
   <!-- va include elementele sugerate obținute
        după invocarea asincronă a serviciului Web -->
   <div id="listaSortimente"></div>
</form>
```

Inițial, marcatorul * * cu *id="listaSortimente"* va fi ascuns, fiind completat de către Atlas la obținerea listei de sortimente întoarse de serviciul Web. Elementului *<input id="sortiment" type="text" />* va trebui să-i atașăm comportamentul de autocompletare, stabilind de asemenea și metoda serviciului Web ce va avea în responsabilitate returnarea listei de sugestii.

Vom adopta metoda declarativă, la finalul paginii inserând liniile de mai jos:

```
<script type="text/xml-script">
  <page xmlns:script=
     "http://schemas.microsoft.com/xml-script/2005">
     <components>
        <textBox id="sortiment">
           <behaviors>
```

```
                <!-- se defineşte modul de
                     autocompletare, prin apelul unei
                     metode a unui serviciu Web -->
                <autoComplete
                   completionList="listaSortimente"
                   serviceURL="Portocale.asmx"
                   serviceMethod="furnizeazaSortimente"
                   minimumPrefixLength="1"
                   completionSetCount="15"
                   completionInterval="500" />
             </behaviors>
          </textBox>
       </components>
    </page>
</script>
```

Se poate deduce ușor faptul că pentru controlul de tip text cu *id="sortiment"* se asociază un marcator XHTML ce va conține rezultatul metodei serviciului Web invocat, unde lungimea prefixului trebuie să fie minimum 1 (această valoare o va avea parametrul *prefixText* de mai sus), mulțimea elementelor sugerate va fi de maxim 15 (parametrul *count* va avea asociată această valoare) și intervalul de completare va fi de 500 de milisecunde.

Maniera efectivă de transfer asincron al datelor între client și server este realizată în mod transparent de către Atlas. În captura-ecran de mai jos pot fi urmărite informațiile despre sortimentele sugerate la introducerea literei „A" de către utilizator, plus apelurile AJAX dintre pagină și serviciul Web. Puteți remarca faptul că datele vehiculate sunt codificate în stilul JSON (în cazul nostru, parametrii de intrare sunt perechi nume-valoare incluse într-o listă, iar rezultatul obținut reprezintă un tablou de șiruri de caractere).

Din moment ce vom selecta unul dintre sortimente, acesta va fi automat introdus ca text în cadrul câmpului formularului Web.

Dorim să adăugăm și un buton etichetat „Alege", la apăsarea căruia să fie invocat – tot în manieră asincronă – un alt serviciu Web, care să realizeze o anumită acțiune (în acest caz, va întoarce un șir de caractere ce va fi redat fără a se realiza reîncărcarea întregului document). Astfel, putem reîmprospăta diverse zone ale paginii, fie în mod explicit, fie periodic sau în funcție de alți factori.

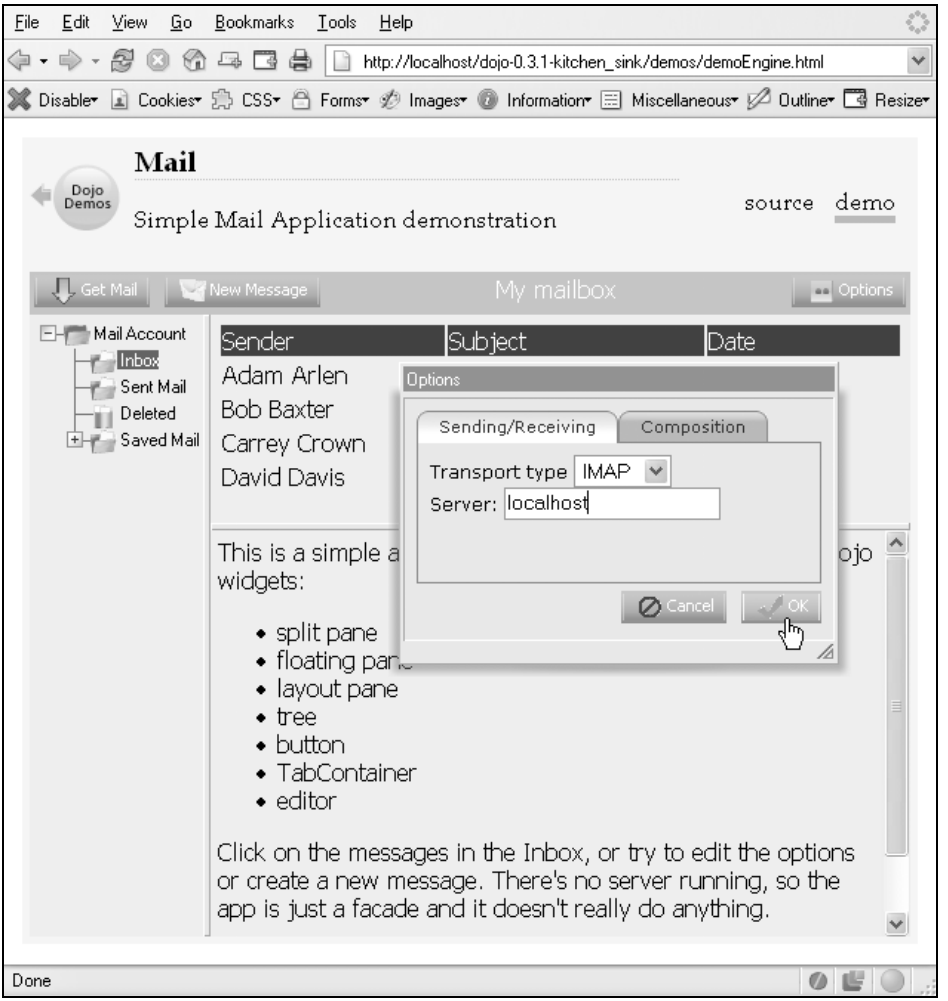


Figura 34. *Folosirea unui control de tip* autoComplete *în Atlas ASP.NET*

Pentru aceasta, va trebui să mai concepem un serviciu Web – *CantitPortocale.asmx* – al cărui cod C# simplu e dat în continuare (omitem construcțiile neesențiale):

```
// afişează sortimentul preluat de la client
[WebMethod]
public String listeazaSortiment (string query) {
  // preluăm parametrul
  string sortiment = Server.HtmlEncode(query);
  // dacă nu există sau e vid,
  // întoarcem un mesaj de avertizare
  if (!String.IsNullOrEmpty(sortiment)) {
    return String.Format("Aţi ales sortimentul {0}!",
      sortiment);
  }
```

```
    else {
      return "Parametrul de intrare primit e vid...";
    }
}
```

Este disponibilă o singură metodă care preia valoarea unui control ASP.NET via obiectul *Server* și întoarce un șir de caractere. Desigur, mai util pentru utilizator ar fi fost să obțină stocul de portocale din respectivul sortiment; acesta este însă un exercițiu ușor de implementat de către cititor.

La formularul descris mai sus, vom adăuga următoarele:

```
<!-- buton ce invocă un alt serviciu -->
<input id="alege" type="button" value="Alege"
   onclick="javascript:Invoca()" />
<div id="rezultat"></div>
```

După cum se remarcă, la apăsarea butonului va fi invocată metoda *listeazaSortiment()*, de data aceasta via JavaScript. Rezultatul va fi conținut de un element <*div*>. Funcția *Invoca()* are liniile de mai jos:

```
function Invoca() {
  // preluăm sortimentul
  var sortim = document.getElementById("sortiment");
  // invocăm metoda
  Portocale.CantitPortocale.listeazaSortiment
    (sortim.value, TrateazaTransfer);
}
function TrateazaTransfer (rezultat) {
  // inserăm rezultatul în <div>
  document.getElementById("rezultat").innerHTML =
    rezultat;
}
```

Cu *getElementById()* oferită de DOM preluăm numele sortimentului și invocăm metoda serviciului Web – se folosește forma *SpatiuDeNume.ServiciuWeb.Metoda()*. Primul argument este tocmai parametrul de intrare al metodei, iar al doilea desemnează funcția JavaScript ce va fi apelată la terminarea transferului. Nu facem decât să inserăm în marcatorul <*div*> rezultatul obținut de la server.

Ca mediul Atlas să poată realiza managementul transferului de date, în antetul paginii trebuie să introducem o referință la serviciul Web invocat:

```
<!-- face referință la serviciul Web invocat
     prin JavaScript în stilul AJAX -->
<atlas:ScriptManager runat="server" id="Atlas">
   <services>
```

```
        <atlas:servicereference
            path="~/CantitPortocale.asmx"
            type="text/javascript" />
    </services>
</atlas:ScriptManager>
```

La rulare, vom obține un rezultat precum cel din figura 35.

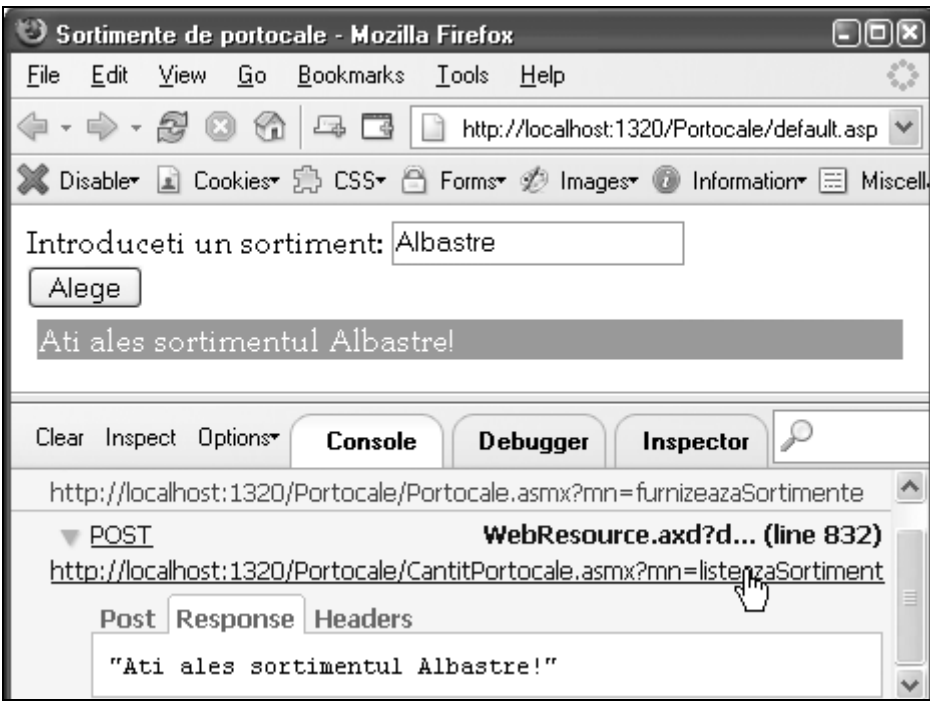


Figura 35. *Actualizarea conținutului fără reîncărcarea întregului document*

Alte abordări AJAX pentru .NET sunt *AJAX.NET*, *MagicAjax.NET*, *JSON.NET* și *xap4net*.

Cele descrise mai sus pot fi realizate și în Java, prin intermediul *framework*-ului DWR (*Direct Web Remoting*) sau via *jMaki*, *SWATO* ori *Taconite*. De asemenea, Google pune la dispoziție GWT (*Google Web Toolkit*), scopul acestui instrument fiind cel de a oferi suport pentru scrierea în Java de aplicații orientate spre AJAX. Pentru PHP, se poate recurge la pachetul *HTML::AJAX*, disponibil în colecția PEAR, iar programatorii Perl pot utiliza extensiile *Mason* pentru AJAX.

Alte cadre de lucru oferă suport pentru AJAX în mai multe limbaje de programare, de menționat fiind CPAINT (*Cross-Platform Asynchronous Interface Toolkit*), *JSON-RPC* și *Sajax*. Nu trebuie uitat nici *Ruby on Rails*, unul dintre

framework-urile de dezvoltare a aplicațiilor Web pe partea de server foarte cunoscute în acest moment, cu suport pentru AJAX.

În funcție de preferințe, de cerințele problemei, de avantaje și/sau de platformă, dezvoltatorul poate adopta una dintre soluțiile disponibile.

Referințe

Alboaie, L, Buraga, S., „Dialoguri despre SOAP", *NET Report*, februarie 2003: http://www.infoiasi.ro/~busaco/publications/articles/SOAP.pdf

Asleson, R., Schutta, N., *Foundations of AJAX*, Apress, 2006

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006: http://www.infoiasi.ro/~busaco/books/xml/

Buraga, S., *Tehnologii Web*, Matrix Rom, București, 2001: http://www.infoiasi.ro/~busaco/books/web.html

Buraga, S., Ciobanu, G., *Atelier de programare în rețele de calculatoare*, Polirom, Iași, 2001: http://www.infoiasi.ro/~lrc/

Buraga, S. *et al.*, *Programare Web în bash și Perl*, Polirom, Iași, 2002: http://www.infoiasi.ro/~cgi/

Buraga, S. (coord.), *Situri Web la cheie. Soluții profesionale de implementare*, Polirom, Iași, 2004: http://www.infoiasi.ro/~busaco/books/webapps/

Buraga, S. (coord.), *Tendințe actuale în proiectarea și dezvoltarea aplicațiilor Web*, Matrix Rom, București, 2006: http://www.infoiasi.ro/~busaco/books/nw05/

Dospinescu, O., *Dezvoltarea aplicațiilor în Visual Basic .NET*, Polirom, Iași, 2004

Eernisse, M., *A Simpler Ajax Path*, OnLamp.com, 2005: http://www.onlamp.com/pub/a/onlamp/2005/05/19/xmlhttprequest.html

Kulchenko, P., Ray, R., *Programming Web Services with Perl*, O'Reilly, 2002

Mueller, J., *Mining Google Web Services: Building Applications with the Google API*, Sybex, 2004

Pereira, S., *Developer Notes for prototype.js*: http://www.sergiopereira.com/articles/prototype.js.html

Perry, B., *AJAX Hacks*, O'Reilly, 2006

Powell, T., Schneider, F., *JavaScript 2.0: The Complete Reference* (Second Edition), McGraw-Hill/Osborne, 2004

Swart, B., „Delphi 6 Web Services: SOAP", *The Delphi Magazine*, Issue 72, August 2001: http://www.thedelphimagazine.com/Samples/1273/article.html

Tanasă, Ș., Olaru, C., *Dezvoltarea aplicațiilor Web folosind Java*, Polirom, 2005

Wood, K., *Delphi Developer's Guide to XML*, Wordware Publishing, 2001

Zakas, N., McPeak, J., Fawcett, J., *Professional AJAX*, Wrox Press, 2006

Zametti, F. *et al.*, *Survey of AJAX/JavaScript Libraries*, OSAF.com, 2005: http://wiki.osafoundation.org/bin/view/Projects/AjaxLibraries

* * *, *AHAH (Asynchronous HTML and HTTP)*: http://microformats.org/wiki/rest/ahah

* * *, *Ajaxian*: http://www.ajaxian.com/
* * *, *AJAX Components for Java*: http://blueprints.dev.java.net/ajaxcomponents.html
* * *, *Apache2Triad*: http://apache2triad.sourceforge.net/
* * *, *Apache Axis2*: http://ws.apache.org/axis2/
* * *, *Apache Geronimo*: http://geronimo.apache.org/
* * *, *Apache XML Activity*: http://xml.apache.org/
* * *, *Atlas ASP.NET*: http://atlas.asp.net/
* * *, *Biblioteca NuSOAP*: http://dietrich.ganx4.com/nusoap/
* * *, *CPAN (Comprehensive Perl Archive Network)*: http://cpan.perl.com/
* * *, *Developer*: http://www.developer.com/
* * *, *Dojo*: http://dojotoolkit.org/
* * *, *DOM-Drag*: http://www.youngpup.net/2001/domdrag/
* * *, *DWR (Direct Web Remoting)* : http://getahead.ltd.uk/dwr/
* * *, *ECMA (European Computer Manufacturers Association)*: http://www.ecma-international.org/
* * *, *Extensia de depanare Firebug*: http://www.joehewitt.com/software/firebug/
* * *, *Google API*: http://www.google.com/apis/
* * *, *Google AJAX Search API*: http://code.google.com/apis/ajaxsearch/
* * *, *GWT (Google Web Toolkit)*: http://code.google.com/webtoolkit/
* * *, *Instrumentul gSOAP*: http://gsoap2.sourceforge.net/
* * *, *Mason*: http://masonhq.com/
* * *, *Microsoft Visual Studio Express*: http://msdn.microsoft.com/vstudio/express/
* * *, *Modulul SOAP::Lite*: http://www.soaplite.com/
* * *, *Modulul SOAP::MIME*: http://sourceforge.net/projects/soapmime
* * *, *MSDN (Microsoft Developer Network)*: http://msdn.microsoft.com/
* * *, *O'Reilly OnLAMP*: http://www.onlamp.com/
* * *, *PEAR (PHP Extensions and Add-ons Repository)*: http://pear.php.net/
* * *, *Portal dedicat serviciilor Web*: http://www.webservices.org/
* * *, *Proiectul jMaki*: https://ajax.dev.java.net/
* * *, *Sajax*: http://www.modernmethod.com/sajax/
* * *, *Situl bibliotecii Prototype*: http://prototype.conio.net/
* * *, *Situl oficial PHP*: http://www.php.net/
* * *, *World Wide Web Consortium, Boston*, 2006: http://www.w3.org/
* * *, *XML.com*: http://www.xml.com/

Capitolul 7

Retrospectivă și perspective

„Realitatea trebuie înțeleasă numai prin spargerea ei în părți cât mai mici."

(René Descartes)

1. Procesul de dezvoltare a serviciilor Web

Recapitulând cele prezentate în capitolele anterioare, procesul de dezvoltare a serviciilor Web implică trei categorii de activități (a se urmări și figura 1):

– *oferirea de servicii*, ceea ce presupune dezvoltarea propriu-zisă a serviciilor puse la dispoziție într-un mediu de tip *enterprise*, plus expunerea unei documentații (descrieri) privitoare la interfața serviciilor; implementarea poate fi realizată pe baza limbajelor de programare, cadrelor de lucru și platformelor actuale, iar descrierile sunt specificate via WSDL și alte metode adiționale;
– *organizarea (catalogarea) serviciilor Web* conform unor criterii vizând tipul de afacere modelată, localizarea ofertantului, industria considerată etc; în acest sens, una dintre inițiativele majore este reprezentată de UDDI; prin intermediul UDDI, un ofertant de servicii poate fi ușor contactat de către posibilii solicitanți (clienți);
– *solicitarea de servicii* fie direct, atunci când interfața acestora este deja cunoscută, fie indirect via UDDI; solicitantul își va proiecta aplicația dorită și va invoca funcționalitățile serviciilor conform documentului WSDL asociat fiecărui serviciu în parte ori apelând la o interfață de programare (API) specifică. Schimbul de date între serviciu și client se va realiza – în mod uzual – prin intermediul SOAP, folosindu-se un protocol de transport oferit de Internet (cel mai frecvent, se recurge la HTTP). O alternativă la SOAP este dată de modelul REST, în care mesajele interschimbate, marcate de asemenea în XML, sunt transferate direct prin HTTP, fără a fi încapsulate în „plicuri".

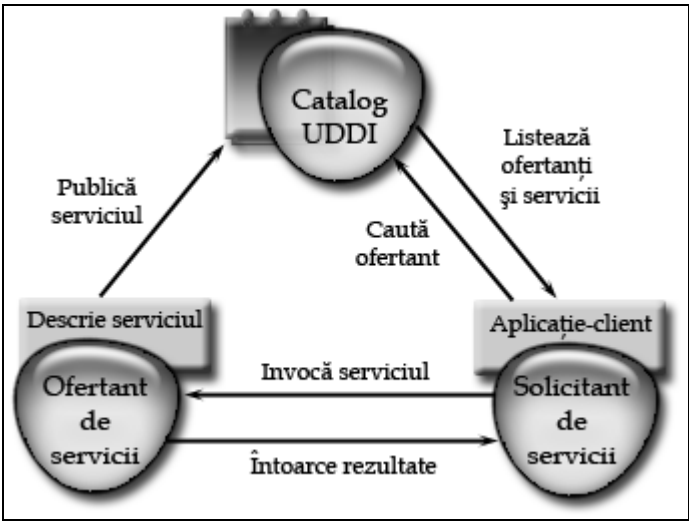


Figura 1. *Activitățile desfășurate în dezvoltarea serviciilor Web*

Acest proces are loc în cadrul unui mediu orientat spre servicii, respectându-se principiile arhitecturale SOA (*Service-Oriented Architecture*) pe care le-am prezentat în cadrul primului capitol al acestei lucrări.

O aplicație Web proiectată pe baza SOA poate fi compusă din diverse straturi funcționale, conform celor ilustrate de figura 2.

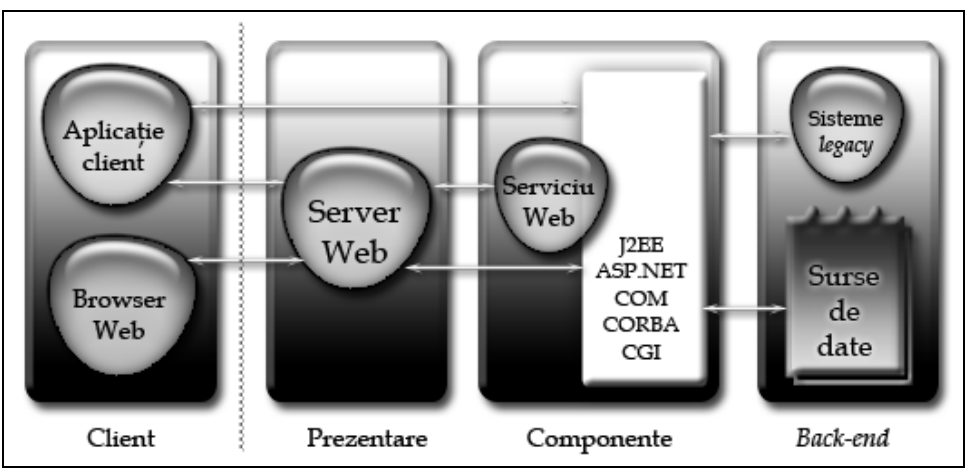


Figura 2. *Structura aplicațiilor software axate pe SOA*

2. Alte inițiative vizând serviciile Web

2.1. *Privire de ansamblu*

În ceea ce privește dezvoltarea de aplicații SOA în general și de servicii Web în particular, tehnologiile SOAP, WSDL și UDDI nu sunt singulare, actualmente fiind disponibile o serie de specificații și inițiative adiționale privitoare la:

1. modul de adresare: *WS-Addressing*;
2. inspecție și descoperire: *WS-Inspection*, *WS-Discovery*;
3. mesagerie: *WS-ReliableMessaging*, *WS Attachments*, *WS-Routing* etc.;
4. securitate și autorizare: *WS-Security*, *WS-Trust*, *WS-Policy* și altele;
5. procesarea tranzacțiilor: *WS-Coordination*, *WS-Transaction*;
6. relația cu interfața-utilizator: WSRP (*Web Services for Remote Portlets*), WSIA (*Web Services for Interactive Applications*);
7. asigurarea inter-operabilității și fluxului de activități (*workflow*): BPEL (*Business Process Execution Language*) și *WS-Choreography*.

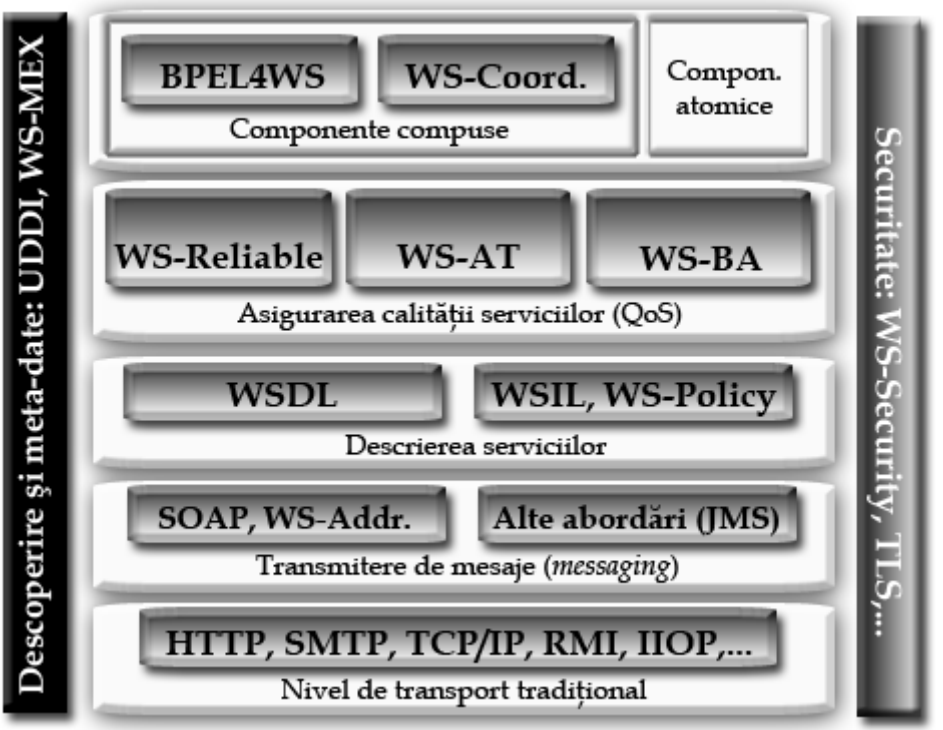


Figura 3. *Nivelurile de specificații adiționale privitoare la serviciile Web*

Trebuie, de asemenea, menționată existența unei suite de specificații complementare, concurente, care se intitulează *ebXML* și care a fost în vogă în perioada 2002-2005. Aceste specificații au fost date ulterior spre standardizare organizației OASIS, semnatara multor documente privitoare la diverse aspecte legate de SOA și serviciile Web.

Vom descrie în continuare o parte dintre direcțiile de evoluție a serviciilor Web, cititorii interesați putând aprofunda aceste aspecte prin consultarea resurselor bibliografice disponibile.

2.2. *Adresarea serviciilor prin* WS-Addressing

Inițiativa *WS-Addressing* specifică modul de utilizare a unor mecanisme – independente de protocolul de transport folosit – destinate adresării serviciilor și mesajelor. Specificația, propusă de IBM, definește elementele XML menite a identifica punctele terminale ale serviciilor Web și maniera de asigurare a securității comunicațiilor dintre aceste puncte terminale. Sistemele de mesagerie (*messaging systems*) pot realiza transportul mesajelor în cadrul unor rețele de comunicații – uzual, folosite în mediul *enterprise* – care pot include noduri de procesare a datelor, precum *firewall*-uri sau porți (*gateways*), indiferent de protocoalele disponibile la nivel scăzut.

Din perspectiva *WS-Addressing*, modelul de descriere a serviciilor Web (*i.e.*, WSDL) este extins prin introducerea conceptului de referință la punctele terminale (*endpoint reference*). Astfel, maniera în care se specifică adresa unui serviciu Web este mai fină, deoarece nu implică doar un URI (de exemplu, pot fi specificate meta-date suplimentare și adresele mai multor documente WSDL).

Mesajele vehiculate pot fi identificate și corelate prin anteturile *MessageID* și *RelatesTo*, ceea ce deschide perspective interesante privind tranzacțiile și fluxurile de date. Sunt introduse și atribute precum *To*, *From*, *ReplyTo* și *FaultTo*, care pot fi folosite de către agenții implicați în comunicare pentru a procesa un mesaj și răspunsurile aferente la acesta.

2.3. *Descoperirea serviciilor*

În capitolul 5 am discutat deja despre maniera de descoperire a serviciilor fie în manieră centralizată via UDDI, fie descentralizat prin *WS-Inspection*.

În cadrul acestei secțiuni, vom prezenta succint modul de descoperire dinamică, specificat de *WS-Discovery* (propunere a corporației Microsoft). În loc de a se stoca informațiile privitoare la servicii într-un catalog central, modelul *WS-Discovery* adoptă strategia folosită și de sistemele *wireless*: un serviciu Web își

poate anunța – via mesaje *multicast* – prezența, atunci când este „atașat" infrastructurii Internet. Un procedeu similar are loc atunci când serviciul „părăsește" rețeaua. Rețeaua este una constituită *ad hoc*, compusă din entități interesate de serviciul respectiv (*i.e.*, clienți care intenționează să invoce anumite operații). Limitarea traficului de mesaje poate avea loc prin constituirea unui așa-numit *proxy* de descoperire (*discovery proxy*) care este implementat tot ca un serviciu Web.

În afară de mesajele de tip *Hello* (anunț al prezenței unui serviciu) și *Bye* („plecarea" serviciului), sunt specificate și următoarele două operații:

– *Probe* – încearcă să detecteze un anumit serviciu ori mulțime de servicii prin transmiterea unui mesaj (*multicast*) către un grup de calculatoare, răspunsul fiind expediat direct solicitantului. Dacă rețeaua posedă un *proxy* de descoperire, atunci metoda *Probe* va fi executată direct de către acesta;
– *Resolve* – intenționează să „rezolve" un serviciu, căutarea efectuându-se după numele acestuia, trimițându-se un mesaj în întreaga rețea. Serviciul care se potrivește cu numele dat, va răspunde solicitantului. Acest proces este similar celui folosit de protocolul ARP (*Address Resolution Protocol*).

Un exemplu de mesaj SOAP care încapsulează o cerere *Probe* este furnizat mai jos (pentru localizare se folosește protocolul LDAP – *Lightweight Directory Access Protocol*):

```
<env:Envelope>
  <env:Header>
     <wsa:Action>
http://schemas.xmlsoap.org/ws/2004/10/discovery/Probe
     </wsa:Action>
     <wsa:To>
     urn:schemas-xmlsoap-org:ws:2004:10:discovery
     </wsa:To>
  </env:Header>
  <env:Body>
     <wsd:Probe>
       <!-- dorim asocierea serviciului Web -->
       <wsd:Types>sfa:Binder</wsd:Types>
       <!-- ...din cadrul domeniului info.uaic.ro -->
       <wsd:Scope MatchBy=
"http://schemas.xmlsoap.org/ws/2004/10/discovery/ldap">
ldap:///ou=info,o=uaic,c=ro
       </wsd:Scope>
     </wsd:Probe>
  </env:Body>
</env:Envelope>
```

Pe baza acestei inițiative și a altora înrudite (precum *WS-Eventing*), poate fi constituită o infrastructură de descoperire dinamică sofisticată, oferind suport pentru managementul serviciilor Web. Suplimentar, în conjuncție cu UPnP (*Universal Plug and Play*), serviciile Web pot rula pe diverse tipuri de dispozitive, fiind regăsite în mod dinamic. De altfel, există deja disponibil un ghid privitor la implementarea serviciilor Web pe calculatoare cu resurse limitate, precizându-se un profil minimal denumit WS-DP (*Device Profile for Web Services*).

2.4. *Coordonarea serviciilor Web*

Deoarece serviciile Web pot fi incluse într-o arhitectură distribuită complexă, se pune problema *colaborării* între diversele entități software implicate pentru a rezolva o anumită problemă specifică (de exemplu, oferirea unei soluții de petrecere a vacanței de către un sit de tip *e-travel*).

Orice colaborare presupune acceptarea unui contract (*agreement*). Pentru sistemele bazate pe transfer de mesaje – cazul serviciilor Web –, un contract, în forma cea mai simplă, presupune respectarea unui format comun de date (*e.g.*, specificat printr-o schemă XML), a unui stil de transmitere a mesajelor (unidirecțional, cerere/răspuns, notificare etc.) și a unei modalități de descoperire a participanților la realizarea problemei. Aceasta implică o activitate de *coordonare*. Cu cât interacțiunile dintre servicii sunt mai sofisticate, cu atât procesul de coordonare devine mai dificil. Modul de coordonare poate fi unul centralizat, implicând existența unui coordonator explicit cu rol de „șef", sau nu.

Inițiativa WS-ReliableMessaging

Unul dintre aspecte importante este cel referitor la realizarea de comunicații fiabile. Necesitatea existenței unui mecanism de asigurare a fiabilității transferului de date provine și din următoarele presupuneri greșite pe care le au dezvoltatorii de aplicații privind un sistem distribuit (formulate de Peter Deutsch la începutul anilor '90 ai secolului trecut):

1. rețeaua este fiabilă (*reliable*);
2. latența datelor este nulă;
3. lățimea de bandă e infinită;
4. rețeaua este sigură (*secure*);
5. topologia rămâne neschimbată;
6. există un singur administrator;
7. costul transportului de date este zero;
8. rețeaua este omogenă.

În contextul serviciilor Web (exemple tipice de componente ale unui sistem distribuit), se poate folosi specificația *WS-ReliableMessaging*, redactată de către IBM. Scopul acestui document este cel de a stabili un protocol care să permită livrarea în siguranță a mesajelor, chiar și în prezența unor erori (*failures*) ale componentelor software, ale platformei utilizate ori ale rețelei.

Pe baza unor informații suplimentare atașate în antetul mesajelor SOAP, datele pot fi grupate în secvențe. Secvența de mesaje este stabilită de către expeditor, iar ordinea trebuie să fie respectată atunci când secvența ajunge la destinatar (nodul *ultimateReceiver*). Partenerii dialogului trebuie să accepte contractul privitor la secvența de mesaje vehiculate. Inițierea unei secvențe are loc prin operația *CreateSequence*, iar partenerul dialogului își va arăta disponibilitatea printr-un răspuns *CreateSequenceResponse*. Această abordare nu necesită prezența unui coordonator explicit. Fiabilitatea este asigurată prin confirmarea fiecărui mesaj în parte via elementele *Sequence*, *SequenceFault*, *SequenceAcknowledgement* și *AckRequested*.

Liniile de mai jos reprezintă confirmarea mesajelor de la 1 la 7 și de la 9 la 10 dintr-o secvență (mesajul 8 nu a fost confirmat încă):

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:rm=
     "http://schemas.xmlsoap.org/ws/2005/02/rm">
   <env:Header>
      ...
     <rm:SequenceAcknowledgment>
       <rm:Identifier>
           ...
       </rm:Identifier>
       <rm:AcknowledgmentRange Upper="7" Lower="1" />
       <rm:AcknowledgmentRange Upper="10" Lower="9" />
     </rm:SequenceAcknowledgment>
   </env:Header>
   <env:Body>
      ...
   </env:Body>
</env:Envelope>
```

Protocolul este neutru în ceea ce privește infrastructura de transport folosită efectiv. Transferul datelor trebuie să respecte diverse precondiții (una fiind, de exemplu, ordinea mesajelor vehiculate). La nivelul HTTP, o inițiativă este HTTPR (*Reliable HTTP*).

Coordonarea explicită a serviciilor Web

O *tranzacție* reprezintă un mecanism care asigură că toți participanții din cadrul unei aplicații vor obține același rezultat, conform unui contract (*agreement*) mutual agreat. În mod tradițional, o tranzacție care are proprietățile de mai jos (întâlnite sub denumirea de ACID – *Atomicity, Consistency, Isolation, Durability*) se numește *atomică*:

– atomicitate – tranzacția se încheie cu succes dacă toate acțiunile pe care aplicația trebuie să se realizeze în cadrul tranzacției au putut avea loc; în caz contrar, tranzacția eșuează și nici una dintre acțiuni nu va fi executată (insuccesul unei singure operații va cauza invalidarea execuției tuturor acțiunilor din cadrul tranzacției);
– consistență – tranzacțiile vor produce întotdeauna rezultate consistente și vor păstra transformările de stare ale aplicației la terminare;
– izolare – stările intermediare ce apar în timpul desfășurării unei tranzacții vor fi invizibile pentru alte tranzacții; tranzacțiile apar ca fiind executate secvențial, chiar dacă sunt realizate în manieră concurentă; aceasta se poate realiza prin „încuierea" (*locking*) resurselor pe durata desfășurării tranzacției, astfel încât nici o altă tranzacție să nu aibă posibilitatea folosirii lor, evitându-se posibilele conflicte de acces;
– durabilitate – după ce o tranzacție se finalizează cu succes, schimbările survenite pot fi gestionate până la apariția unei erori catastrofale.

Un exemplu de tranzacție este cel referitor la oferta unui serviciu de turism, care implică existența unui agent (*travel agent*) având rolul de a rezerva bilete la avion (sau tren, autocar etc.), de a închiria diverse mijloace de transport la destinație, de a rezerva o cameră la un hotel și, eventual, de a pune la dispoziție și alte servicii (*e.g.*, angajarea unui ghid ori programarea unui tur de vizitare a localității respective). Eșuarea uneia dintre activitățile ce compun tranzacția (de exemplu, imposibilitatea găsirii unei soluții de cazare) conduce la abandonarea întregului serviciu.

În contextul serviciilor Web, compunerea unuia sau mai multor servicii conduce la constituirea aplicațiilor tranzacționale ale căror componente sunt slab conectate și distribuite pe sisteme eterogene și independente. Astfel, unele proprietăți ale tranzacțiilor atomice pot fi aplicate, în diverse situații, mai puțin strict.

Intervine necesitatea existenței unui mecanism flexibil de coordonare, care să permită colaborări, fluxuri de lucru (*workflow*), procesare în timp real etc. În

unele situaţii, această coordonare trebuie realizată explicit, interacţiunea având loc uzual între doi participanţi (desigur, putem generaliza la un număr oarecare de părţi componente). Coordonarea se poate desfăşura spontan – aşa cum se întâmplă la un schimb sincronizat de mesaje de tip cerere/răspuns (invocarea unui serviciu şi obţinerea rezultatului) – sau printr-un coordonator. Pentru această a doua situaţie, a fost propusă specificaţia *WS-Coordination*, având drept iniţiatoare companiile BEA, IBM şi Microsoft. Se precizează un cadru de lucru extensibil care să ofere protocoale pentru coordonarea acţiunilor desfăşurate de aplicaţiile distribuite.

Specificaţia introduce conceptul de *context de coordonare* (*Coordination Context*), care apare într-un bloc antet SOAP şi care identifică unic o activitate ce trebuie desfăşurată în comun. Pentru a iniţia o activitate executată de mai multe servicii, un serviciu Web va expedia un context de coordonare la serviciile vizate. Contextul va conţine informaţiile necesare pentru ca o entitate să decidă dacă doreşte să ia parte la activitatea preconizată sau să refuze propunerea. Aceste informaţii depind de specificul activităţii. Mulţimea tipurilor de coordonare este deschisă, noile tipuri putând fi definite de o anumită implementare, atât timp cât toţi participanţii le cunosc şi consimt să le folosească. Comportamentul unui anumit serviciu este expus prin intermediul unui proces de înregistrare (*registration*) transmis coordonatorului. Pe baza înregistrării, pentru a rezolva o problemă dată poate fi invocat un întreg grup de servicii.

De asemenea, serviciul coordonator are astfel posibilitatea să orchestreze serviciile Web (vezi secţiunea 2.9) şi/sau să realizeze tranzacţii.

Interacţiunile de tip B2B (*Business-to-Business*) se pot desfăşura conform următoarelor protocoale definite de *WS-Coordination*:

– *tranzacţie atomică* (*atomic transaction*) – corespunde tranzacţiilor cu proprietăţile ACID menţionate mai sus şi e specificată de *WS-AtomicTransaction* (vezi *infra*);
– *intra-enterprise* – în cadrul aceleaşi organizaţii (domeniu), tranzacţiile au loc atomic de-a lungul variatelor aplicaţiilor interne;
– *inter-enterprise* – reprezintă activitatea de bază desfăşurată în context B2B, necesitând un grad mare de inter-operabilitate cu sistemele tranzacţionale existente;
– *activitate de afaceri* (*business activity*) – necesită o flexibilitate sporită a componentelor sistemului faţă de cerinţele stricte ale tranzacţiilor atomice (ACID). Interacţiunile pot avea loc timp îndelungat, iar serviciile sunt de cele mai multe ori slab conectate. Restricţiile privind atomicitatea şi izolarea sunt relaxate. Acest model are legătură şi cu maniera de realizare a proceselor de afaceri.

Realizarea tranzacțiilor atomice

Pentru specificarea tranzacțiilor atomice în contextul serviciilor Web, există disponibilă inițiativa *WS-AtomicTransaction*.

Inițierea unei procesări de date (văzută ca o unitate atomică) are loc prin protocolul numit *Completion*. Un serviciu Web înregistrat pentru *Completion* are abilitatea de a înștiința coordonatorul atunci când inițiază o tranzacție. De asemenea, acest protocol definește forma mesajelor ce pot fi folosite pentru a comunica rezultatul final al tranzacției inițiatorului acesteia.

Specificația *WS-AtomicTransaction* pune, de asemenea, la dispoziție și protocolul numit *Two-Phase Commit*. Toți participanții înregistrați trebuie să ia în comun decizia de a duce până la capăt (*commit*) o tranzacție ori de a o abandona. Suplimentar, protocolul asigură faptul că toți participanții vor fi anunțați asupra rezultatului final. Sunt definite două variante ale protocolului: una volatilă și cealaltă durabilă. Protocolul volatil poate fi folosit de participanți care dețin resurse volatile (*e.g.*, *cache*), pe când cel durabil este destinat participanților având resurse considerate durabile (precum baze de date ori fișiere).

Inter-operabilitatea este asigurată grație folosirii protocolului SOAP. Politicile de efectuare a tranzacțiilor pot fi specificate prin intermediul *WS-Policy*, iar aspectele legate de securitate sunt lăsate în seama propunerilor *WS-Security* și *WS-Trust*, pe care le vom prezenta mai jos.

În conjuncție cu *WS-BusinessActivity*, pot fi implementate tranzacții care se desfășoară timp îndelungat (de exemplu, procesări complexe).

Managementul cozilor de mesaje

Deseori, mai ales în contextul transmiterii asincrone de mesaje, apare necesitatea utilizării cozilor (*queues*) de tranzacții durabile. Comunicarea între entitățile participante are loc via cozi de mesaje (MQ – *Message Queue*). Aplicația care expediază datele le plasează – în maniera unei tranzacții atomice (via *WS-AtomicTransaction*) – într-o coadă. Mesajul se consideră stocat cu succes în cadrul cozii doar dacă nu apare nici o excepție de procesare (adică tranzacția nu a eșuat).

Subsistemul de management al cozilor va avea misiunea de livrare a mesajului emis de expeditor către destinatarul final. Acest proces poate fi desfășurat la un moment de timp diferit de cel al includerii mesajului în coadă. De asemenea, locația cozii de mesaje nu trebuie să coincidă cu cea a expeditorului ori a destinatarului.

Aplicaţia care preia mesajul din cadrul cozii va realiza acest lucru printr-o tranzacţie atomică, astfel încât datele vor fi livrate cu succes (şi eliminate din coadă) doar în cazul în care nu apar excepţii de procesare la nivelul destinatarului.

2.5. *Accesarea resurselor serviciilor Web*

În unele situaţii, schimbul de mesaje între un serviciu şi clienţii săi presupune un dialog mai complex decât o simplă pereche cerere-răspuns. De exemplu, pot apărea cazuri în care este nevoie de specificarea transferului unor interogări de baze de date, fluxuri de informaţii multimedia, liste de enumerare a unor valori etc.

Deoarece HTTP este un protocol lipsit de stări (*stateless*), nu se poate cunoaşte dacă diverse cereri succesive provin de la acelaşi client. Apare necesitatea de a prezerva anumite date de-a lungul mai multor accesări înrudite ale unui serviciu Web. Acest aspect se realizează prin intermediul sesiunilor, exact ca şi în cazul aplicaţiilor Web convenţionale.

Pentru enumerarea datelor via mesaje succesive, se va stabili o sesiune prin operaţia *Enumerate*, precizată de specificaţia *WS-Enumeration*. De asemenea, acest document defineşte şi modul de accesare a secvenţelor de date. Sesiunea este declarată la nivel abstract prin intermediul unui context de enumerare (*enumeration context*), care reprezintă un cursor logic ce parcurge o secvenţă de informaţii oferite de o anumită sursă de date (sistem de baze de date relaţionale, server de baze de date native XML etc.). Aceste date, reprezentate via *XML Infoset*, vor fi livrate printr-o succesiune de mesaje SOAP.

Cea mai simplă operaţie este *Pull*, care permite ca o sursă de date, într-un anumit context de enumerare, să producă o secvenţă de elemente XML (reprezentate intern prin *Infoset*) încapsulate în corpul unui mesaj SOAP. Fiecare operaţie *Pull* ce va urma va întoarce următoarele *N* elemente din cadrul secvenţei.

Conform *WS-Enumeration*, se pun la dispoziţie şi operaţiile:

– *Renew* – utilizată să extindă validitatea unui context de enumerare (reîmprospătează o sesiune);
– *GetStatus* – oferă informaţii privitoare la un context de enumerare;
– *Release* – eliberează resursele deţinute în cadrul unei sesiuni şi elimină un context de enumerare.

În caz de eroare (enumerarea s-a terminat neprevăzut), sursa de date va transmite un mesaj de tip *EnumerationEnd*.

În conjuncţie cu specificaţia prezentată mai sus, poate fi folosită *WS-Transfer*, care are în responsabilitate definirea operaţiilor de bază privitoare la managementul entităţilor de date vehiculate via servicii Web.

WS-Transfer definește *resursa* ca fiind orice entitate adresabilă de o referință a unui punct terminal, entitate ce oferă o reprezentare XML a datelor. De asemenea, se introduce conceptul de fabrică (*factory*): un serviciu Web care poate „construi" o resursă având la dispoziție reprezentarea ei XML.

Sunt specificate operații privitoare la crearea, actualizarea, obținerea și ștergerea resurselor (operații de tip CRUD – *Create, Retrieve, Update, Delete*). Crearea unei resurse este realizată de o fabrică, aceasta constituind resursa dorită și stabilind reprezentarea sa inițială.

Ilustrăm în continuare modalitatea de creare a unei resurse privitoare la achiziția unui sortiment de portocale:

```
<env:Envelope>
 <env:Header>
  <!-- acțiunea realizată -->
  <wsa:Action>
  http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
  </wsa:Action>
  ...
 </env:Header>
 <env:Body>
  <p:portocale p:xmlns="urn:portocale.info">
    <p:achizitii>
      <p:tip>Portocale grecești fără coajă</p:tip>
      <p:cod>P10-01-GR</p:cod>
      <p:cantit p:um="kg">3374</p:cantit>
    </p:achizitii>
  </p:portocale>
 </env:Body>
</env:Envelope>
```

Dacă transferul de date se desfășoară în mod asincron, posibilele evenimente, ce pot surveni pot fi tratate prin intermediul mecanismului oferit de *WS-Eventing*. Un serviciu Web (numit *subscriber*) își precizează lista evenimentelor asupra cărora este interesat (și va fi notificat), evenimente generate de alt serviciu Web (denumit *event source*). Sunt precizate și operațiile ce pot fi efectuate asupra evenimentelor sau grupurilor de evenimente. De asemenea, pot exista și entități de tip *broker* de evenimente cu rolul de a agrega ori redistribui notificări de evenimente.

La finalul acestei secțiuni, trebuie să amintim și inițiativa *WS-Management*, care oferă diverse facilități referitoare la managementul serviciilor Web (inclusiv al resurselor transferate sau al evenimentelor de comunicare) pe baza protocolului SOAP.

2.6. *Accesarea meta-datelor asociate serviciilor*

Pentru a facilita dezvoltarea și exploatarea serviciilor Web, fiecărui astfel de serviciu i se pot asocia, prin intermediul meta-datelor, diverse descrieri adiționale. Meta-datele sunt specificate într-un format ușor de procesat de către calculator, oferind suport pentru automatizarea activităților de implementare sau de regăsire a serviciilor. De asemenea, meta-datele pot fi de folos pentru creșterea gradului de inter-operabilitate a serviciilor.

Meta-datele descriu maniera schimbului de mesaje, funcționalitățile și cerințele (*requirements*) asociate unui serviciu. Aceste cerințe formează așa-numita politică (*policy*) de acces. Formatele de date și șabloanele de interschimb de mesaje sunt specificate în cadrul documentelor WSDL, după cum am văzut în capitolul 2. Politicile de acces sunt subiectul specificației *WS-Policy*.

Întreg setul de meta-date formează așa-numitul contract asociat serviciului, contract exprimat prin construcții (elemente) *XML Schema*, WSDL și *WS-Policy* (a se vedea și figura 4).

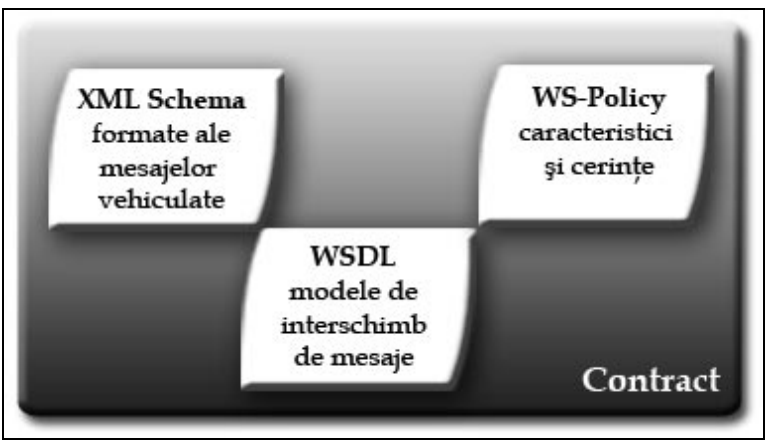


Figura 4. *Categoriile de meta-date care formează contractul asociat unui serviciu Web*

Documentele WSDL nu sunt capabile să ofere posibilelor entități de calcul sau dezvoltatorilor de aplicații informații privitoare la diverse caracteristici operaționale, de securitate sau referitoare la exploatarea serviciilor Web. De exemplu, ar fi util de cunoscut dacă accesarea serviciului se realizează pe baza unui sistem de autentificare obligatorie, ori dacă serviciul e disponibil doar între anumite intervale de timp. Toate aceste meta-date pot fi exprimate via *WS-Policy* pe baza unui set extensibil de aserțiuni. Dacă WSDL se axează pe unele descrieri funcționale asociate serviciilor Web, construcțiile *WS-Policy* permit atașarea de

descrieri non-funcționale sau privitoare la asigurarea calității serviciilor. Alte aspecte, precum cele referitoare la semantica serviciilor Web, vor putea fi modelate de către alte prevederi (*e.g.*, OWL-S). De asemenea, setul de politici exprimate de *WS-Policy* nu vizează doar punctele terminale, așa cum se întâmplă în cazul WSDL.

Oferim drept exemplificare următorul fragment de document XML care precizează că un serviciu Web nu va fi disponibil între orele 10 și 11 în ziua de duminică a fiecărei săptămâni (politica efectivă se exprimă printr-un vocabular XML definit de noi, aparținând spațiului de nume „*s*”):

```
<wsp:Policy>
   <wsp:All>
      <s:PlanificareRevizie>
         <s:Oprire s:ziua="Duminica"
                   s:start="1000" s:final="1100" />
      </s:PlanificareRevizie>
   </wsp:All>
</wsp:Policy>
```

Diversele politici de acces pot fi atașate serviciilor prin intermediul WS-PA (*WS-PolicyAttachment*). Maniera de interschimb de meta-date este precizată de specificația WS-MEX (*WS-MetadataExchange*), meta-datele putând fi specificate în diverse așa-numite dialecte (*XML Schema*, WSDL, *WS-Policy* etc.). Un exemplu de cerere de furnizare a meta-datelor privitoare la politicile de acces este următorul:

```
<wsx:GetMetadata>
  <wsx:Dialect>
     http://schemas.xmlsoap.org/ws/2002/12/policy
  </wsx:Dialect>
</wsx:GetMetadata>
```

2.7. *Securitatea serviciilor Web*

Aspecte generale privind securitatea

Securitatea joacă un rol esențial în dezvoltarea serviciilor Web. În mod general, aspectele privind securitatea datelor iau în considerație confidențialitatea, autentificarea, autorizarea, integritatea, nerepudierea, intimitatea (*privacy*) și disponibilitatea.

Confidențialitatea se referă la imposibilitatea unei terțe entități să aibă acces la datele vehiculate între doi receptori. O soluție ar fi aceea a constituirii unei conexiuni private între cele două puncte terminale ale canalului de comunicație,

datele circulând printr-un tunel oferit de o *rețea privată virtuală* (VPN – *Virtual Private Network*). A doua soluție este oferită de *criptarea datelor* prin intermediul unei suite de tehnici, implementate de biblioteci specializate și/sau oferite de mediile de dezvoltare existente.

În unele situații, precum cele din domeniul *e-business*, accesul la serviciul Web nu trebuie să fie public, acordat oricui. *Autentificarea* presupune un mecanism care permite utilizatorilor să acceseze un serviciu după verificarea identității utilizatorului (în general, pe baza unui nume de cont și a unei parole furnizate de acel utilizator). De cele mai multe ori, serverul Web oferă suport pentru autentificări de bază sau bazate pe algoritmi de tip *digest*, precum MD5. În cazul serviciilor Web, există însă inițiative speciale, precum SAML (*Security Assertion Markup Language*) – vezi *infra*.

Autorizarea este asociată autentificării și specifică acțiunile (rolurile) pe care un utilizator poate să le realizeze. Un utilizator autentificat poate să nu fie în mod implicit și autorizat să aibă acces la o anumită resursă a sistemului. Sistemele de autorizare permit administratorului să definească politici pentru controlul accesului la servicii. În mod tradițional, se folosesc drepturi de acces (permisiuni) și liste de control al accesului (ACL – *Access Control List*), utilizatorii fiind clasificați pe grupuri sau roluri. Prin intermediul controlului accesului bazat pe roluri (RBAC – *Role-Based Access Control*), sistemul de management al securității poate fi mai ușor mulat pe structura organizației. În ceea ce privește utilizatorul, în mod uzual se folosește o tehnică de tip SSO (*Single Sign-On*), bazat pe existența unui singur procedeu de autentificare comună pentru accesul la mai multe resurse (situri, servicii etc.).

Tehnicile de tip *digest* pot asigura și *integritatea* informațiilor, fiind posibilă detectarea oricărei încercări malițioase de modificare a datelor transmise. De asemenea, tot pentru realizarea integrității, pot fi folosite *semnături digitale* (a nu se confunda însă cu semnăturile electronice care se referă la totalitatea mijloacelor prin care o informație poate „purta" semnătura proprietarului ei). Semnăturile digitale pot fi stocate în format XML – și vehiculate, desigur, via mesaje SOAP – prin intermediul specificației *XML Signature*.

Nerepudierea asigură faptul că expeditorul mesajului nu poate afirma că nu l-a trimis. Pentru aceasta se folosesc *certificate digitale*, care stochează datele privind identitatea unei entități deținătoare a unui secret (o parolă, o serie a unei cărți de credit, a certificatului digital în sine etc.). Certificate digitale sunt disponibile într-un format precum X.500. Un certificat digital este emis de o autoritate de certificare (CA – *Certification Authority*), iar verificarea certificatelor se realizează de către o autoritate de înregistrare a acestora (RA – *Registration Authority*). Cataloagele de certificate, împreună cu autoritățile de certificare (CA) și înregistrare

(RA) formează așa-numita *infrastructură cu chei publice* (PKI – *Public Key Infrastructure*). Oferirea de servicii PKI în contextul spațiului Web se poate realiza via XKMS (*XML Key Management Specification*).

Disponibilitatea impune ca o anumită resursă să poată fi accesată la momentul oportun. Un serviciu Web indisponibil clienților săi pur și simplu nu există, ceea ce are repercusiuni negative mai ales în contextul afacerilor electronice. Printre cauzele care pot eroda disponibilitatea se enumeră atacurile de refuz al serviciilor – DoS (*Denial Of Service*) sau DDoS (*Distributed DoS*). Atacurile distribuite sunt deosebit de periculoase pentru asigurarea disponibilității serviciilor Web publice.

Intimitatea este deseori confundată cu confidențialitatea. Confidențialitatea presupune imposibilitatea ca datele aflate în tranzit să fie accesate de persoane rău-voitoare, pe când intimitatea vizează drepturile ce trebuie respectate privind caracterul (subiectul) datelor vehiculate. Multe dintre vulnerabilitățile raportate în domeniul comerțului electronic sunt de fapt breșe în sistemul de asigurare a intimității utilizatorilor. Pentru datele transportate pe Internet sunt adoptate în mod unanim diverse tehnici de criptare, dar la nivel de server ar putea să fie stocate necorespunzător (de exemplu, în clar) în cadrul unui sistem de stocare vulnerabil. Un atacator va avea astfel acces la seria cărții de credit a unui client, pe baza unei conexiuni directe cu serverul de baze de date sau a unui tehnici de tip XSS (*Cross-Site Scripting*). Alte probleme vizând pierderea intimității sunt cauzate de configurarea necorespunzătoare a sistemelor – detalii și în cartea lui Sabin Buraga, *Proiectarea siturilor Web* (ediția a doua), Polirom, Iași, 2005.

Niveluri de securitate

Vom face mai jos o scurtă trecere în revistă a cerințelor de securitate pe care trebuie să le îndeplinească serviciile Web.

Construirea unor servicii Web sigure (din punctul de vedere al securității) trebuie să ia în considerare aspecte ca:

– realizarea unui model (*Web Services model*) care are la bază o analiză detaliată a factorilor care amenință comunicarea și punctele terminale (*endpoints*) ale serviciilor Web;
– existența unui cadru de lucru (*framework*) facilitând securizarea resurselor, ușor integrabil într-o arhitectură orientată spre servicii;
– asigurarea unui mecanism de autentificare și autorizare a părților care comunică în cadrul sistemului constituit.

Majoritatea scenariilor de securitate sunt legate de securitate la nivel de rețea/transport, securitate la nivel de date și integritate la nivel de mesaj.

La nivel de rețea, mijlocul de protecție generală împotriva incidentelor de securitate se concretizează în așa-numitul zid de protecție (*firewall*). Deoarece serviciile Web folosesc în mod nativ protocolul HTTP, mesajele vehiculate vor putea traversa Internetul fără aportul filtrelor impuse de *firewall*-uri (acesta este, de fapt, și unul dintre avantajele serviciilor Web față de tehnologiile CORBA sau DCOM). Totuși, *firewall*-urile vor fi de folos pentru stăvilirea altor tipuri de atacuri, de nivel inferior (*e.g.*, inundări cu pachete). De asemenea, la nivel de rețea poate fi folosit protocolul IPSec (*IP Secure*).

În ceea ce privește nivelul de transport, se recurge la SSL (*Secure Sockets Layer*) și la versiunea mai recentă, standardizată, TLS (*Transport Layer Security*). Aceste protocoale sunt folosite pentru autentificarea și confidențialitatea mesajelor HTTP. Autentificarea se realizează via certificate digitale, iar confidențialitatea prin criptare. Datele HTTP securizate via SSL/TLS fac obiectul specificației HTTPS, portul folosit fiind 443, în loc de cel implicit – 80.

La nivel de aplicație, în primul rând se poate utiliza S/MIME (*Secure MIME*), soluție pentru asigurarea confidențialității, integrității și nerepudierii în cazul mesajelor de poștă electronică. Dacă se adoptă o metodă de transfer SOAP peste SMTP, atunci S/MIME poate fi atractiv și în cazul serviciilor Web.

Pentru HTTP sau alte protocoale (*e.g.*, Jabber), un prim deziderat este să se ofere o securitate persistentă a mesajelor SOAP transportate. Tehnologiile amintite mai sus vizează o securitate exterioară. Includerea de informații privind securitatea în cadrul unui mesaj SOAP este subiectul specificației *WS-Security*, actualmente standard OASIS. *WS-Security* definește metode speciale de inserare a datelor criptate sau a semnăturilor digitale în anteturile SOAP. În plus, se oferă suport pentru includerea de jetoane (*tokens*) de securitate arbitrare. Alte detalii vor fi furnizate mai târziu în cadrul acestui capitol.

Securitatea în contextul XML

Confidențialitatea este asigurată via *XML Encryption*, specificație a Consorțiului Web, care oferă mijloacele pentru criptarea unor porțiuni din documentele XML și stocarea datelor criptate în format XML. *XML Encryption* poate fi benefic în contextul serviciilor Web, datorită abilității criptării doar a unor fragmente XML. Mesajele SOAP pot include informații care trebuie să rămână publice (privitoare la dirijare, de exemplu), dar pot conține și părți criptate. Standardul *XML Encryption* nu introduce noi algoritmi de criptare, putând fi folosiți cei deja consacrați, ci doar specifică anumite meta-date referitoare la aceștia și la modul de utilizare într-un context dat.

Un exemplu de criptare a informațiilor privitoare la cartea de credit a unui utilizator este următorul (numele clientului e public, dar celelalte informații private sunt criptate – în acest caz, se folosesc și semnături digitale):

```
<?xml version='1.0'?>
<plata xmlns='http://portocale.info/Plati'>
   <nume>Portocal Escu</nume>
   <EncryptedData Id='EData'
      xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <EncryptionMethod Algorithm=
      'http://www.w3.org/2001/04/xmlenc#aes128-cbc'/>
      <ds:KeyInfo xmlns:ds=
         'http://www.w3.org/2000/09/xmldsig#'>
         <ds:RetrievalMethod URI='#EKey' Type=
       "http://www.w3.org/2001/04/xmlenc#EncryptedKey">
            <ds:KeyName>SymmetricKey</ds:KeyName>
        </ds:RetrivalMethod>
      </ds:KeyInfo>
      <CipherData>
         <CipherValue>pj3io2sa25fF</CipherValue>
      </CipherData>
   </EncryptedData>
</plata>
```

Pentru rezolvarea integrității, avem la dispoziție *XML Signature*, specificație comună a Consorțiului Web și a IETF (*Internet Engineering Task Force*). Ca mai sus, se oferă atât posibilitatea semnării digitale a unor porțiuni dintr-un document XML, cât și exprimarea în XML a informațiilor referitoare la semnăturile digitale. Modul de integrare a elementelor *XML Signature* în cadrul mesajelor SOAP este detaliat de specificația *WS-Security*.

Problemele de autentificare și autorizare în contextul serviciilor Web se încearcă a fi rezolvate de o suită de tehnologii. Diverse declarații privitoare la anumite aspecte legate de securitate pot fi exprimate cu ajutorul SAML (*Security Assertions Markup Language*), standardizat de OASIS. Limbajul SAML poate fi de folos și pentru stocarea unor informații privitoare la utilizatorul final (*e.g.*, precizarea unei limite inferioare a unui depozit bancar în cazul unui serviciu de creditare). SAML este utilizat mai ales pentru a specifica informații referitoare la autentificare sau autorizare care s-au întâmplat în trecut, permițând ca aceste informații să fie plasate într-un mesaj SOAP (de exemplu, „persoana *P* a fost autorizată conform unor politici de acces *A* la momentul *M*"). Dacă destinatarul acelui mesaj are încredere în aserțiunea (*assertion*) semnalată prin SAML, poate permite utilizatorului să aibă acces la serviciul Web dorit.

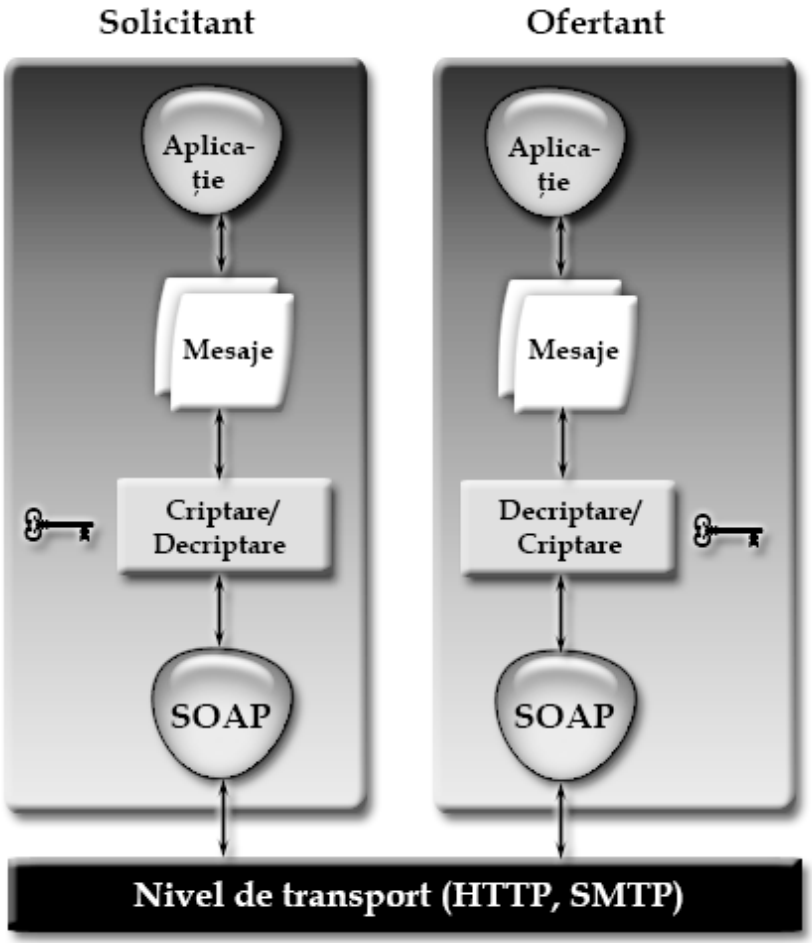


Figura 5. *Procesul de criptare/descriptare a mesajelor SOAP*

Drept exemplificare, furnizăm următorul fragment de document, care precizează că autentificarea utilizatorului *blue.ory* s-a realizat cu succes, pe baza metodei de autentificare oferită de sistemul Kerberos:

```
<saml:AuthenticationStatement
   AuthenticationInstant="2006-09-01T10:33:33Z"
   AuthenticationMethod="Kerberos">
   <saml:Subject>
      <saml:NameIdentifier
         NameQualifier="verisign.com/ams">
         blue.ory
      </saml:NameIdentifier>
   </saml:Subject>
</saml:AuthenticationStatement>
```

Suplimentar, pentru a exprima regulile de control al accesului în termenii XML, putem recurge la XACML (*Extensible Access Control Markup Language*). Decizia de autorizare modelată printr-o aserțiune SAML poate fi bazată pe regulile stipulate via XACML. Regulile pot desemna politici de acces la resurse (citire, scriere, execuție etc.) bazate pe diverse caracteristici ale entității în cauză (*e.g.*, „lista clienților poate fi parcursă doar de membrii departamentului de vânzări") sau ale protocolului („invocarea serviciului Web se poate realiza numai via HTTPS"). De asemenea, XACML poate fi folosit și la managementul drepturilor digitale – DRM (*Digital Rights Management*).

Specificația XKMS (*XML Key Management Specification*) oferă suportul pentru gestiunea cheilor publice via XML. Astfel, în mod natural, serviciile PKI – recunoscute a fi dificil de implementat – pot fi puse la dispoziție mai facil, grație serviciilor Web. Recomandarea XKMS presupune existența a două servicii Web:

– X-KRSS (*XML Key Registration Service Specification*) trebuie să ofere suport pentru managementul ciclului de viață al acreditărilor (*credentials*) asociate cheilor publice;
– X-KISS (*XML Key Information Service Specification*) expune metode de interogare pentru obținerea și validarea cheilor publice.

Protocolul XKMS este în strânsă legătură cu SOAP, fiind de tip cerere--răspuns. De asemenea, se oferă facilități pentru procesări asincrone și formularea de cereri compuse. Via XKMS, complexitatea virează de la client la intrastructură, codul de implementat fiind mult mai simplu.

Securizarea serviciilor Web. Tehnologii și standarde

Tehnologiile prezentate mai sus nu se focalizează în principal asupra securității serviciilor Web, ci iau în considerație și securitatea datelor XML.

Pentru securizarea mesajelor SOAP a fost redactat un set de specificații purtând numele *WS-Security*, efort comun al corporațiilor IBM și Microsoft, actualmente standardizat de OASIS. Aspectele de securitate nu mai vizează în mod obligatoriu protocolul folosit (*e.g.*, HTTP cu TLS), ci se situează la nivelul mesajelor, în termeni de asigurare a integrității, confidențialității și autentificării. *WS-Security* include și alte specificații, precum *WS-Trust* (pentru asigurarea încrederii), *WS-SecurityPolicy* (privitoare la politici de acces sigur) și *WS-Privacy* (pentru asigurarea intimității). La un nivel superior, se află *WS-SecureConversation* (având în vedere siguranța conversațiilor între serviciile Web), *WS-Federation* (vizând gruparea serviciilor într-o federație partajând aceleași politici de securitate) și *WS-Authorization* (oferind suport pentru metode de autorizare sofisticate).



Figura 6. *Diversele aspecte privind securitatea serviciilor Web sunt specificate pe niveluri*

WS-Trust definește modul de stabilire a relațiilor de încredere într-un sistem. Aceste relații pot avea loc direct sau prin intermediari (*brokeri*). Intermedierea relațiilor de încredere se realizează printr-un *proxy* de încredere (*trust proxy*). De asemenea, se permit mecanisme pentru delegare (*delegation*) și impersonalizare (*impersonation*).

Prevederile *WS-SecurityPolicy* pot fi utilizate pentru formularea unor politici de securitate pe baza limbajului *WS-Policy*.

WS-Privacy folosește *WS-SecurityPolicy*, *WS-Security* și *WS-Trust* pentru a realiza schimburi de mesaje conform unor reguli de asigurare a intimității.

Specificația *WS-SecureConversation* permite ca un solicitant și un ofertant de servicii Web să stabilească un context de autentificare mutuală prin mesaje SOAP.

WS-Federation duce securitatea la nivel de federație de servicii Web, unde „federație" presupune inter-operabilitatea diverselor strategii și specificații de securitate folosite de o suită de servicii Web, care formează un domeniu de încredere. *WS-Federation* poate fi folosit și în contextul EASI (*Enterprise Application Security Integration*).

WS-Authorization poate fi considerat complementar XACML, descriind maniera de specificare și de management al politicilor de acces la serviciile Web.

Din punct de vedere tehnic, *WS-Security* presupune existența într-un mesaj SOAP a unui bloc antet *Security* care trebuie să aibă prevăzut atributul *mustUnderstand* cu valoarea *true*:

```
<Envelope>
   <Header>
      ...
     <Security role="http://www.portocale.info/Plati"
        mustUnderstand="true">
```

```
        ...
      </Security>
        ...
    </Header>
        ...
</Envelope>
```

Dacă un nod SOAP este incapabil să proceseze blocul privitor la securitatea mesajului, atunci mesajul va fi distrus. Deoarece un mesaj poate include mai multe blocuri *Security*, vom indica prin atributul *role* cărui serviciu Web îi sunt adresate informațiile de securitate din fiecare bloc în parte. Conținutul propriu-zis al elementului *Security* va consta din jetoane (*tokens*) de securitate, precum *UsernameToken* (referitor la autentificare prin SAML ori alte tehnici) și *BinarySecurityToken* (privitor la diverse date criptate, exprimate cu *XML Signature*, *XML Encryption* sau altfel).

De asemenea, se precizează și diverse scenarii de manipulare a erorilor survenite via elementul *Fault* oferit de SOAP.

Actualmente, unul dintre cele mai versatile instrumente care suportă *WS-Security* este extensia WSE (*Web Services Enhancements*) versiunea 3.0 pentru *.NET Framework*. De exemplu, criptarea unui mesaj SOAP prin folosirea unui certificat digital în format X.509 (care trebuie să se găsească pe fiecare calculator al clientului ce solicită o informație confidențială de la serviciul Web) poate fi realizată astfel:

```
X509SecurityToken certificat = FurnizeazaCertificat();
// adăugăm jetonul privitor la certificat
serviciu.RequestSoapContext.Security.Tokens.Add
  (certificat);
// inserăm un element EncryptedData
serviciu.RequestSoapContext.Security.Elements.Add(
  new EncryptedData(certificat));
```

Menționăm că WSE nu oferă doar facilități în ceea ce privește securitatea serviciilor Web, ci și pentru adresare (*WS-Addressing*) și atașare de alte informații la mesajele SOAP (*WS-Attachments*). Sunt puse la dispoziție și diverse instrumente utile ce pot fi integrate fără probleme în *Visual Studio .NET*. De asemenea, WSE facilitează exploatarea serviciilor Web create în .NET și prin intermediul unor protocoale diferite de HTTP.

2.8. *Asigurarea inter-operabilităţii*

Inter-operabilitatea vizează existenţa unei suite de tehnici şi instrumente care permit aplicaţiilor software (în cazul nostru, serviciile Web şi clienţii aferenţi) să comunice indiferent de tehnologiile şi platformele folosite. Mai mult decât atât, aplicaţiile pot partaja datele indiferent de manierele de reprezentare, de acces şi de tehnologia folosită la momentul rulării pentru managementul acestor date. De asemenea, inter-operabilitatea presupune posibilitatea formării unei platforme virtuale de calcul, pe baza sistemelor de operare, mediilor de rulare (*runtime*) şi componentelor software trecute, prezente şi viitoare.

Tehnologiile actuale menite a oferi sprijin pentru realizarea inter-operabilităţii sunt WSIT (*Web Services Interoperability Technology*), JWSDP (*Java Web Services Developer Pack*), WSE (*Web Services Enhancements*) şi WCF (*Windows Communication Foundation*). La momentul scrierii acestui material, JWSDP se află la versiunea 2. După cum am menţionat în secţiunea anterioară, WSE reprezintă o extensie oferită de Microsoft pentru dezvoltarea de servicii Web securizate şi inter-operabile pe platforma .NET, actualmente fiind disponibilă versiunea 3.0. WSE are la bază specificaţii standardizate, precum *WS-Security*, *WS-Addressing*, *WS-Policy* şi *WS-I*. WCF – cunoscut anterior şi sub numele *Indigo* – desemnează una dintre componentele fundamentale ale noului sistem Windows Vista, menită a oferi o infrastructură pentru comunicarea prin mesaje între diverse componente.

O iniţiativă importantă de standardizare referitoare la inter-operabilitatea serviciilor Web este WS-I (*Web Service Interoperability*), detalii fiind disponibile la *http://www.ws-i.org*.

Principalele avantaje ale inter-operabilităţii sunt cele privind asigurarea independenţei de platformă, evoluţia bazată pe standarde deschise şi utilizarea familiei XML. Printre slăbiciuni se numără problemele de performanţă, dificultatea gestionării per ansamblu a stării sistemului, problemele cauzate de incompatibilitatea datelor (*data mismatch*) din cauza interpretărilor diferite ale standardelor implementate de diverşi ofertanţi. De asemenea, nu întotdeauna poate fi folosit modelul cerere/răspuns, mai ales în cazul unor aplicaţii care trebuie să fie executate în timp-real sau care necesită transferuri de fluxuri de informaţii (*streaming*).

Aspectele importante pe care trebuie să le avem în vedere când concepem servicii Web care să asigure o inter-operabilitate solidă se referă la:

– tipurile de date şi schemele XML proiectate şi folosite în cadrul documentelor WSDL (se recomandă ca, înainte de implementarea propriu-zisă a serviciului, să se realizeze o analiză şi proiectare corespunzătoare a schemelor XML

privitoare la mesajele vehiculate între serviciile Web și posibilii lor clienți; de asemenea, o atenție deosebită trebuie acordată documentului WSDL de descriere a serviciului, deoarece este unicul mijloc prin care un dezvoltator extern are acces la funcționalitățile serviciului în cauză);
– alinierea la standardele în vigoare (mai ales la WS-I, prezentat mai jos);
– controlul versiunilor (*versioning*);
– compatibilitatea cu variantele anterioare ale instrumentelor folosite (de exemplu, JWSDP 1.x/2.x *versus* WSE 2.0/3.0 sau WCF).

Etapa de proiectare vizează alegerea tipurilor de date, scrierea schemelor XML și stabilirea modului de tratare a erorilor. Primul pas este cel de definire a schemei XML, construcțiile folosite fiind cât mai simple posibil (a se evita aspectele avansate privitoare la reprezentarea datelor via *XML Schema*). Tot aici, trebuie să avem în vedere profilele de inter-operabilitate WS-I. Nu trebuie să presupunem că tipurile de date vor fi verificate de clienți, că versiunea SOAP folosită de diversele implementări este și va rămâne aceeași, că transferurile de date vor fi realizate doar în manieră sincronă. De asemenea, este posibil ca instrumentele utilizate să nu fie compatibile între ele.

În faza de dezvoltare trebuie avute în vedere aspecte privind comunicațiile asincrone, partajarea datelor/sesiunilor, mecanismul de mesagerie folosit și securitatea.

La momentul exploatării în practică, principalele probleme sunt cele legate de performanță și scalabilitate.

Inter-operabilitatea poate fi asigurată și prin interpunerea unei componente de tip *wrapper*, care transformă mesajele clientului în formatul înțeles de către serviciu și invers. Acest *wrapper* poate fi situat la nivel de server, la nivel de client sau într-un strat intermediar. Există situații în care se folosesc două componente *wrapper*, una pe mașina clientului, cealaltă pe server. Soluțiile care permit realizarea acestui „pod" (*bridging*) de conectare a două aplicații incompatibile folosesc fie tehnologiile Java, fie .NET (via *.NET Remoting*).

Alternativ, inter-operabilitatea poate fi obținută grație sistemelor *messaging*, folosind punți de interconectare, adaptori, mesaje SOAP ori sisteme de mesagerie clasică SMTP/POP/IMAP. În acest caz, comunicațiile pot fi și asincrone.

Pentru aplicații compuse (*composite applications*), Sun propune o arhitectură denumită „magistrala de servicii *enterprise*" – ESB (*Enterprise Service Bus*), constituită din componente pentru managementul și orchestrarea sofisticată a tuturor activităților din cadrul unei organizații, asigurându-se inclusiv inter-operabilitatea. În viziunea Sun, ESB reprezintă o etapă evolutivă superioară SOA, oferindu-se suport pentru existența componentelor hibride, a unui management centralizat

şi a unor entităţi *middleware* orientate spre mesaje. De asemenea, prin ESB se înlesneşte asigurarea scalabilităţii.

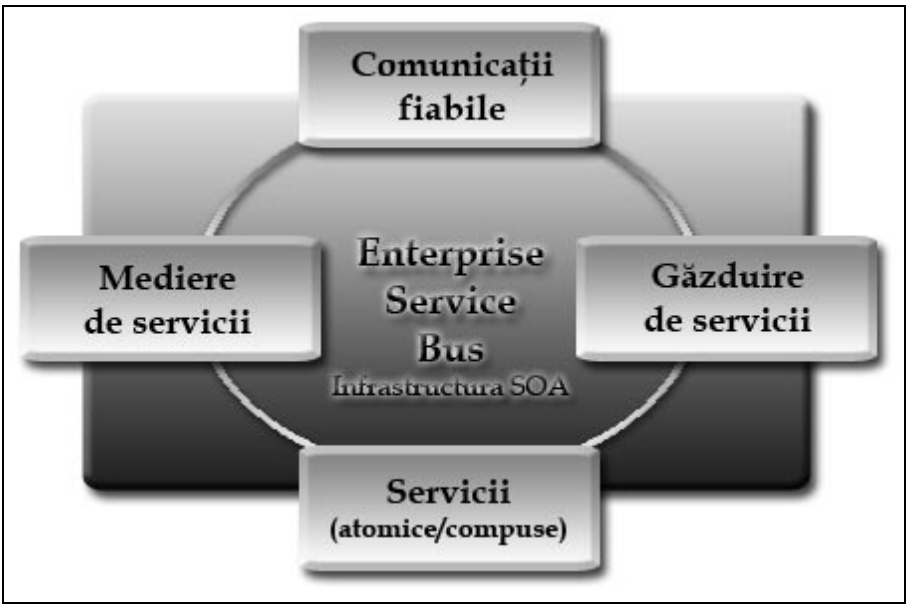


Figura 7. *Componentele ESB*

În ce priveşte aplicaţiile compozite, ele sunt constituite din servicii reutilizabile, putând fi create uşor, frecvent şi la cerere, conform cerinţelor afacerilor întreprinse. Aspectele statice, mai riguroase, ale aplicaţiei sunt separate de cele dinamice, în evoluţie, conforme cu preferinţele utilizatorilor. Dacă Java poate fi limbajul folosit la implementarea aplicaţiilor, compunerea serviciilor poate fi realizată prin BPEL4WS; a se urmări cele discutate în secţiunea de mai jos.

Aliniate arhitecturii ESB, sunt disponibile atât produse comerciale – oferite de BEA, Cape Clear, IBM, Oracle ori Sun –, cât şi aplicaţii *open source* – de exemplu, *Celtix* (IONA), *FUSE* (LogicBlase), *Open ESB* (Sun), *Synapse* (Apache) şi *ServiceMix* (Apache).

Direcţiile de viitor vizează – ori, de fapt, visează? – unificarea platformelor J2EE (*Java 2 Enterprise Edition*) şi .NET. Astfel, pot fi adoptate extensii J2EE care să permită aplicaţiilor .NET să ruleze pe un anumit server de aplicaţii Java. De asemenea, poate fi constituit un strat de translatare la momentul compilării a codului din limbajul intermediar MSIL în secvenţe de *bytecode* Java şi vice-versa. În acest mod, sistemul de operare poate găzdui o unică maşină virtuală capabilă să „înţeleagă" ambele tehnologii (instrucţiuni de nivel scăzut, sistem de tipuri de date, componente etc.). Aplicaţiile .NET pot „consuma" resurse Java, în mod

direct, și invers. Aceste maniere de inter-operabilitate sunt deja suportate de aplicații precum *JBoss*, *Tomcat*, *WebLogic* și *WebSphere*. De asemenea, *Visual MainWin* pentru J2EE reprezintă un *plug-in* Visual Studio care permite dezvoltatorilor să scrie programe C# sau VB.NET (inclusiv servicii Web) care sunt convertite automat după compilare în *bytecode* Java.

Pe o idee oarecum similară se bazează și noua versiune Perl 6, via mașina virtuală Parrot, aflată în plină dezvoltare.

În ceea ce privește Java, noile versiuni J2SE 1.6 și J2EE 1.5 pun un accent pronunțat asupra asigurării inter-operabilității. Un exemplu este și *GlassFish*, disponibil gratuit pentru utilizare, parte a proiectului *Tango*, al cărui scop este oferirea unei suite de tehnologii Web viitoare care să permită inter-operabilitatea dintre aplicațiile Java și cele WCF, pe baza strategiilor WSIT (*Web Services Interoperability Technology*) stipulate de corporația Sun.

Profilul WS-I de bază

Pentru a nivela posibilele diferențe între implementările oferite de diversele medii de dezvoltare a serviciilor Web, organizația WS-I a redactat un prim profil – denumit profilul de bază (*WS-I Basic Profile*) – la care trebuie să se alinieze toate implementările actuale. În prezent este în vigoare versiunea 1.1 a acestui profil. Una dintre prevederile majore ale profilului este aceea a interzicerii codificării în stilul RPC a mesajelor SOAP.

Pentru a indica faptul că o implementare este conformă cu profilul de bază, de cele mai multe ori se recurge la specificatori (meta-date) precum cei dați mai jos (aici, pentru .NET):

```
[WebService]
[WebServiceBinding(
   ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Serviciu : System.Web.Services.WebService
{
   // implementare
}
```

În viitor, WS-I ar putea standardiza și alte profile privitoare la diversele aspecte ale inter-operabilității serviciilor Web.

2.9. *Serviciile Web în contextul proceselor de afaceri*

Este de necontestat faptul că serviciile Web și arhitectura orientată spre servicii au devenit paradigma obișnuită pentru proiectarea și implementarea colaborărilor la nivel de afaceri în cadrul și dincolo de granițele unei organizații. Funcționalitățile

sunt oferite prin intermediul interfețelor standardizate ale serviciilor Web. Serviciile puse la dispoziție de diferite organizații, eventual regăsite grație UDDI, pot fi interconectate în vederea realizării de operații (relativ mai) complexe. Astfel, grupuri de servicii considerate atomice constituie așa-numitele *servicii compozite* (*composite Web services*).

Colaborările inter- sau intra-organizaționale necesită în multe cazuri interacțiuni de durată, conduse de modele explicite de procese de afaceri. Modelarea proceselor de afaceri se regăsește și sub termenii de model al fluxului de activități (*workflow*), orchestrare sau coregrafie.

Există deja o suită de limbaje XML de modelare a proceselor de afaceri, distingându-se WSCI (*Web Service Choreography Interface*), BPML (*Business Process Modeling Language*) și BPEL4WS (*Business Process Execution Language for Web Services*). Aceste abordări iau în considerație atât manierele de comunicare între diversele entități software implicate, cât și *workflow*-urile desfășurate într-un sistem.

Vom discuta în continuare câteva dintre aspectele definitorii ale BPEL4WS. Acest limbaj are ca strămoși WSFL (*Web Services Flow Language*) oferit de IBM și propunerea XLANG inițiată de Microsoft. Dacă primul modela un flux de activități ca un graf orientat specificat de construcții XML, al doilea avea un punct de vedere apropiat de scrierea în pseudo-cod a algoritmilor.

Un *proces* reprezintă orice colecție de activități care conduc la realizarea unui anumit obiectiv (de exemplu, procesarea unei comenzi ori cereri de plată, angajarea unui nou programator într-o companie sau trimiterea unei propuneri de invenție spre aprobare).

În viziunea BPEL4WS, procesele pot fi abstracte sau executabile. Cele abstracte desemnează un protocol specificând tipul mesajelor interschimbate de diverse entități, fără a se oferi detalii privitoare la comportamentul acestora. Procesele executabile definesc:

– o ordine de execuție a activităților constituiente;
– partenerii implicați;
– mesajele vehiculate între parteneri;
– mecanismele de rezolvare a excepțiilor/erorilor.

La rândul ei, o activitate – parte componentă a unui proces – poate fi simplă (*primitive*) sau structurată (*structured*). Tipurile de activități simple sunt următoarele:

– *invoke* – invocarea unei operații oferite de un serviciu Web și descrise de un document WSDL;
– *receive* – așteptarea unui mesaj provenind de la o sursă externă;
– *reply* – trimiterea unui mesaj-replică unei surse externe;

– *wait* – oprirea activității un anumit timp;
– *assign* – copierea datelor dintr-o variabilă în alta;
– *throw* – indicarea erorilor survenite într-o execuție;
– *terminate* – terminarea unei instanțe de serviciu;
– *empty* – desemnează activitatea vidă.

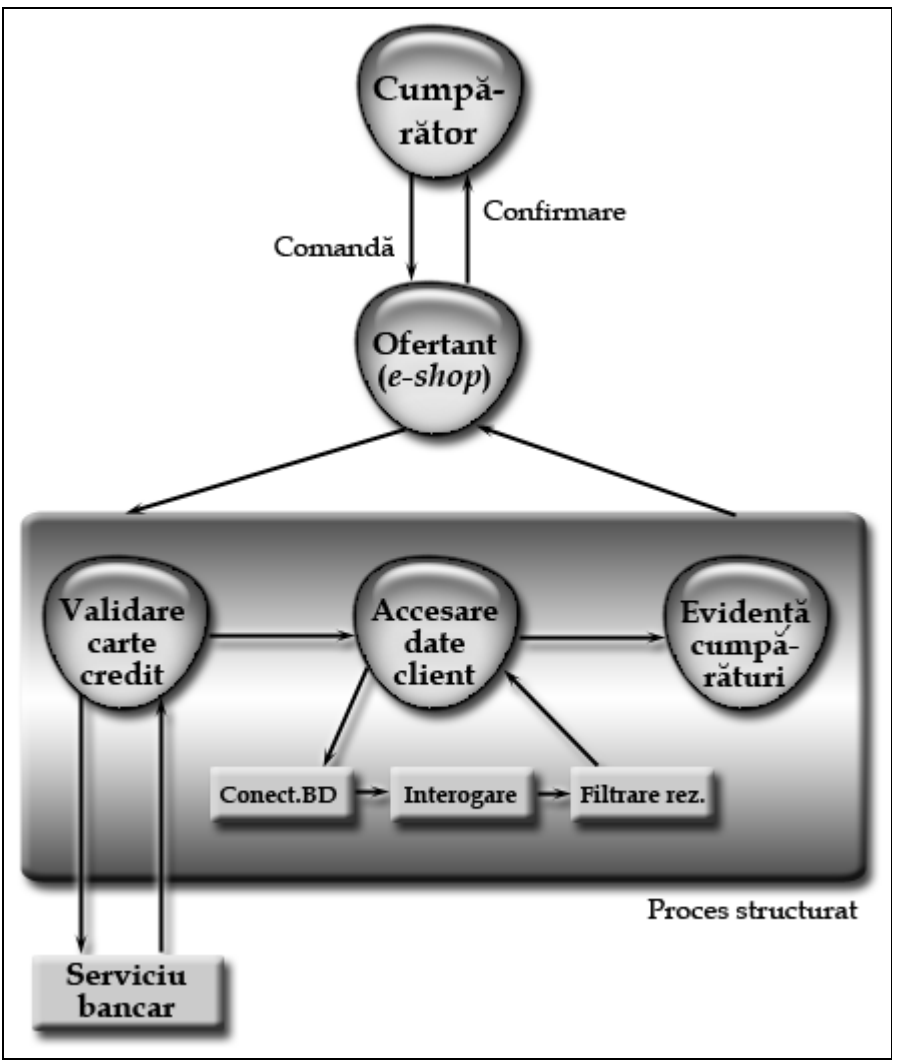


Figura 8. *Reprezentarea grafică a unui* workflow *desfășurat în cazul inițierii unei comenzi din partea unui cumpărător* online *ce dorește să achiziționeze un sortiment de portocale*

Operațiile complexe pot fi modelate prin activități structurate. Astfel, pot fi specificate structuri foarte similare celor puse la dispoziție de limbajele de programare:

– *sequence* – definește o ordine de execuție, secvențială;
– *switch* – desemnează o alternativă (*conditional routing*);
– *while* – permite bucle de execuții;
– *pick* – stabilește condiții de alegere bazate pe evenimente (*e.g.*, *onAlarm*, *onMessage* etc.);
– *flow* – permite execuția în paralel (*parallel routing*);
– *scope* – grupează activități ce vor fi tratate de același *handler* de excepții.

Varianta 2.0, mai recentă, a limbajului introduce noi tipuri de activități (*if-then-else*, *forEach*, *repeatUntil*, *validate* etc.) și transformări ale valorilor variabilelor prin XSLT. De asemenea, este aliniată la prevederile în vigoare privind inter-operabilitatea.

Procesul de verificare a stocului de portocale poate fi modelat astfel (este vorba de o comunicare sincronizată):

```
<process name="verifStocPorto">
   <sequence>
      <!-- dorim să aflăm cantitatea disponibilă
           de portocale -->
      <invoke partner="stocuri"
        inputVariable="portocale"
        outPutVariable="cantit">
      </invoke>
      <flow>
        <!-- alte activități care nu depind
             de 'cantit' -->
        <sequence>...</sequence>
        <!-- obținem cantitatea -->
        <receive partner="stocuri"
          variable="cantit"></receive>
      </flow>
   </sequence>
</process>
```

Desigur, activitățile structurate pot fi imbricate. În cazul activităților care fac parte din aceeași activitate de tip *flow*, ordinea de execuție poate fi controlată via *links* – legături ce permit definirea dependențelor între două activități (activitatea destinație poate începe doar atunci când activitatea sursă s-a încheiat). Astfel se formează un digraf aciclic de activități, cu condiții de tranziție de la o activitate la alta (*transition conditions*) și condiții de unificare (*join conditions*).

Vom exemplifica două dintre șabloanele de realizare a *workflow*-urilor: alegerea exclusivă (*exclusive choice*) și confluența simplă (*simple merge*).

Alegerea exclusivă presupune, pe baza luării unei decizii, alegerea unei singure soluții dintr-o alternativă. De exemplu, managerul unui magazin virtual va fi notificat dacă stocul de portocale este pe sfârșite (cantitatea disponibilă este mai mică de 33 de kilograme), altfel nu. Schematic, aceasta poate fi exprimată prin liniile:

```
<switch>
   <case condition="#cantitMinima">
      <!-- notificăm managerul -->
   </case>
   <case condition="...">
      ...
   </case>
</switch>
```

Confluența simplă precizează un punct din procesul *workflow* în care se reunesc două sau mai multe ramuri de activități alternative, fără a necesita sincronizare (se presupune însă că ramurile alternative nu se execută niciodată în paralel). De exemplu, după ce plata a fost efectuată sau creditul a fost aprobat, produsul comandat poate fi expediat clientului.

Actualmente sunt disponibile diverse sisteme de management al fluxurilor de activități – WFMS (*WorkFlow Management System*) –, majoritatea oferind suport pentru BPEL4WS ori alte propuneri complementare. Ca exemple notabile menționăm instrumentele oferite de BEA, Microsoft, Oracle și Sun, care funcționează pe platforme precum .NET, J2EE sau JBI (*Java Business Integration*).

Ca alternativă la ESB (care oferă suport și pentru *workflow*-uri), corporația Microsoft are în vedere constituirea a ceea ce numește *Windows Workflow Foundation*, integrată cu tehnologiile Microsoft, dar a cărei arhitectură se bazează pe modelul unui procesor (*engine*) BPEL. La nivel de utilizator final, *workflow*-urile pot fi definite grafic direct în Office, iar dezvoltatorii vor avea la dispoziție diverse aplicații precum *Visual Studio Team System*. Actualmente, fluxurile de activități sunt folosite în cadrul serverului *BizTalk* și a noii fundații de servicii de comunicare WCF (*Windows Communication Foundation*).

2.10. *Servicii Web pentru* grid

Orchestrarea și coordonarea sunt esențiale și în cadrul sistemelor de tip *grid*, în care serviciile Web joacă un rol important. Conceptul de *grid* desemnează o entitate oferind servicii de calcul sau acces la date și fiind compusă dintr-o serie de computere privite ca o gazdă Internet unică. Dacă inițial au reprezentat o propunere de infrastructură de calcul distribuit pe scară largă destinată proiectelor științifice și industriale, tehnologiile *grid* au oferit ulterior suport pentru căutarea

și regăsirea informațiilor, indiferent de localizarea lor fizică. Într-un sistem *grid*, serviciile pot fi dinamice și volatile, constituite *ad-hoc*, disponibile pe scară largă și pe termen îndelungat.

O implementare de referință a arhitecturii și protocoalelor *grid* este *Globus*, actualmente ajunsă la versiunea 4. Instrumentele de dezvoltare formează *Globus Toolkit*, disponibile în general pentru sistemele UNIX/Linux. Interfețele de programare formează așa-numitul CoG (*Commodity Grid*) *toolkit*, fiind disponibile pentru limbaje precum Java, Perl ori Python, cu suport și pentru CORBA, .NET și servicii Web. Standardizarea tehnologiilor *grid* este supervizată de *Global Grid Forum*.

În prezent, atenția este focalizată asupra constituirii unei arhitecturi orientate spre servicii *grid*, recurgându-se la tehnologiile Web actuale. Astfel, în cadrul modelul arhitectural OGSA (*Open Grid Service Architecture*) serviciile Web sunt extinse pentru a putea specifica funcționalități de bază și specializate pentru sisteme de tip *grid*. Arhitectura OGSA este compusă din infrastructura OGSI, o suită de interfețe și de modele.

OGSI (*Open Grid Services Infrastructure*) se află la convergența dintre *grid* și serviciile Web, definind mecanismele de bază pentru managementul instanțelor de servicii *grid* (transferul de mesaje, menținerea ciclului de viață etc.). OGSI extinde WSDL pentru a oferi servicii dependente de stare (*stateful*) – utile pentru gestionarea stării globale a sistemului –, moștenirea interfețelor serviciilor, notificarea asincronă a schimbărilor de stare, specificarea referințelor la instanțe de servicii și altele. De asemenea, sunt definite mecanismele prin care clienții pot avea acces la serviciile *grid*-ului.

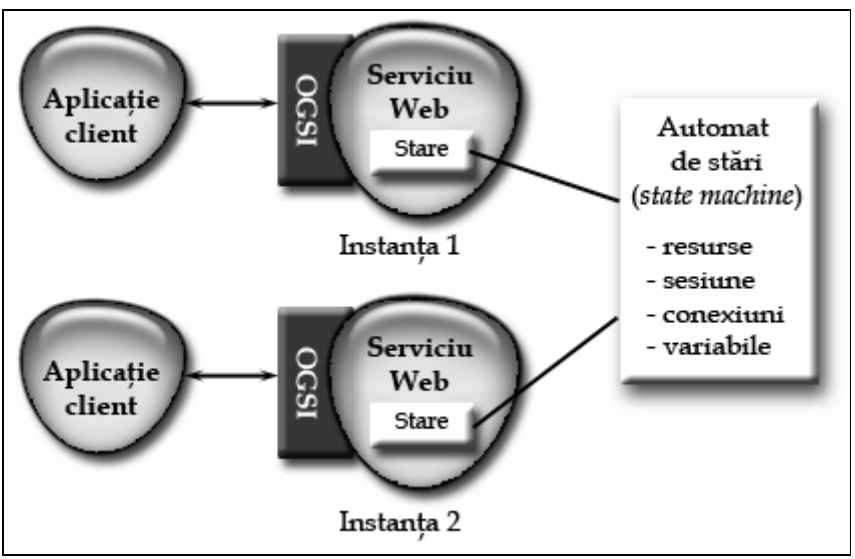


Figura 9. *Menținerea stării fiecărei instanțe de serviciu Web existent într-un* grid

Interfețele (*OGSA Platform Interfaces*) reprezintă diverse servicii (interfețe + comportamente asociate) care nu sunt definite în cadrul OGSI. Se vizează astfel un nivel superior de servicii, precum accesul și integrarea datelor, înregistrarea și managementul resurselor, asigurarea securității, jurnalizarea distribuită, orchestrarea serviciilor etc.

Modelele (*OGSA Platform Models*) sunt o combinație de interfețe și scheme de date utile la reprezentarea entităților – *e.g.*, un server de stocare – care sunt incluse într-un sistem *grid*.

Maniera de atașare a serviciilor la infrastructura de transport Internet, specificarea gazdelor, serviciile specifice unui domeniu de activitate nu fac subiectul specificațiilor OGSA, putând fi definite fie de anumite profile, fie de terțe organizații.

În plus, OGSA definește *WS-Agreement*, prin intermediul căreia se specifică modul în care serviciile *grid* sunt gestionate conform scopurilor unor organizații sau cerințelor aplicațiilor ce trebuie rulate pe *grid*, pe baza unor contracte (*agreements*) prestabilite sau realizate dinamic.

În legătură cu OGSA, trebuie semnalată și inițiativa WSRF (*Web Services Resource Framework*), o familie de specificații vizând accesarea resurselor prin servicii Web, în manieră *stateful*.

O altă direcție este DAIS (*Data Access and Integration Services*) prin care sunt descrise mecanismele de acces la date la nivel înalt, ascunzându-se detaliile privitoare la integrarea datelor stocate în surse multiple, având reprezentări diferite.

Implementările OGSA se axează asupra tehnologiilor Java și .NET. Unul dintre proiectele interesante de dezvoltare este WSRF.NET, bazat pe specificațiile WSRF și implementat în C# sub *.NET Framework*.

Accesul la resursele unui grid poate fi facilitat de existența unui portal, deseori oferind o interfață Web. În acest context, menționăm inițiativele WSRP (*Web Services for Remote Portlets*) și WSIA (*Web Services for Interactive Applications*).

3. SOA în contextul noului Web

Din cele prezentate mai sus, putem deduce că în cadrul SOA pot exista mai multe roluri pe care un dezvoltator Web poate să le joace. În primul rând, cel de implementator de servicii, activitate care presupune crearea și publicarea implementărilor serviciilor rezultate. O a doua ipostază o reprezintă consumatorul de servicii, a cărui misiune – circumscrisă celei a dezvoltatorului de servicii – este de a concepe aplicații-client menite a invoca suite de servicii. Un rol nou îl deține dezvoltatorul de compuneri de servicii care proiectează, generează și publică o

clasă de servicii compuse din alte servicii. Cel de-al patrulea tip vizează asamblarea serviciilor înrudite în vederea exploatării (*deployment*) efective și a managementului operațiilor.

Modelul de creare a aplicațiilor și sistemelor utilizând SOA este descris de o specificație intitulată SCA (*Service Component Architecture*), propusă la finalul anului 2005 de o serie de companii. Conform SCA, se oferă o modalitate neutră – din punctul de vedere al platformei și limbajului de programare – pentru reprezentarea serviciilor care pot fi rulate pe o varietate de medii. Aceste servicii pot fi compuse, implementate de o paletă largă de *framework*-uri și containere (*e.g.*, EJB – *Enterprise Java Beans*). La nivel de implementare, proiectul *Apache Tuscany* oferă suport pentru SCA, în limbajele C++ și Java. De asemenea, poate fi menționat și Eclipse STP (*SOA Tools Project*).

Dacă SOA permite adoptarea unei abordări sistematice de compunere dinamică a serviciilor conform cerințelor industriei, noua direcție de evoluție a spațiului WWW – denumită *Data Web* sau *Web 2.0* – exprimă dorința de existență a unui mecanism flexibil de integrare *ad-hoc* a resurselor, fără constrângeri. Modul de utilizare a Web-ului actual se bazează pe modele de interacțiune bogată (aspect posibil via tehnologii ca RSS, Atom și AJAX) și pe crearea de rețele sociale slab-conectate (a se vedea fenomenul *blogging*, corelat cu existența taxonomiilor create de grupuri de persoane – *folksonomies* – și a aplicațiilor colaborative de tip *wiki*). Web-ul este văzut ca o platformă și nu numai ca depozit de date, în care comunități de utilizatori generează conținut util (pe baza inteligenței colective).

În acest context, apare tipul de aplicație Web hibridă (așa-numitul *mash-up*) care reprezintă un sit Web ori o aplicație Web ce combină în mod agil conținuturi provenite din mai multe surse, în vederea oferirii unei interacțiuni complexe. Sursele sunt în mod uzual *RSS feed*-uri asociate *blog*-urilor sau *wiki*-urilor și servicii Web exploatate prin SOAP ori prin REST. Pentru a facilita această integrare, numeroși ofertanți de servicii Web – începând cu Google și Amazon și continuând cu eBay, Yahoo! și multe alte companii – au pus la dispoziție API-uri specializate.

Mai mult decât atât, există inițiative pentru asocierea de descrieri semantice serviciilor Web, în contextul direcției de evoluție spre *Web-ul semantic* – detalii în lucrările lui Sabin Buraga, *Tehnologii XML*, Polirom, Iași, 2006 și Sabin Buraga, *Semantic Web*, Matrix Rom, București, 2004.

Printre problemele actuale care trebuie rezolvate numără cele legate de neconcordanțe semantice între servicii la nivel de:

– conținut – de exemplu, un serviciu Web care furnizează localizarea unei resurse multimedia (*e.g.*, un film) întoarce adresa numerică (IP) a serverului de stocare, deși solicitantul dorește URI-ul acelei resurse;

– atribut – se solicită tipul MIME video/mpeg, dar serviciul furnizează tipul generic Video;
– unități de măsură – serviciul solicită ca dată de intrare dimensiunea unui fișier audio în octeți, dar i se trimite un număr reprezentând dimensiunea în MB;
– mesaj – clientul solicitant poate furniza rezoluția unei imagini ca pereche lățime-înălțime, însă serviciul Web dorește numărul total de pixeli alocați.

Pentru descrierea semantică și invocarea corespunzătoare a serviciilor Web, eforturile sunt concentrate asupra adoptării unor specificații de nivel înalt, capabile să descrie proprietăți și funcționalități într-o formă procesabilă de calculator. Astfel, s-a propus un cadru de lucru destinat serviciilor Web descrise semantic – SWSF (*Semantic Web Services Framework*), care înglobează o suită de propuneri precum OWL-S și SWSO (*Semantic Web Services Ontology*). O altă inițiativă demnă de interes este WSMO (*Web Service Modeling Ontology*).

Conceptual, un serviciu va fi privit ca un proces, căruia – prin proprietăți – îi vor fi asociate datele de intrare/ieșire și, eventual, diverși parametri de control. Un alt concept important este cel de profil, incluzând meta-date necesare descoperirii, selecției și potrivirii semantice (*matchmaking*) de servicii. Modelul de procesare vede un serviciu ca fiind atomic ori compus, aspect util în ceea ce privește compunerea, execuția și monitorizarea serviciilor Web. Ultima componentă de interes privește infrastructura (*grounding*), desemnând cum poate fi executat serviciul și extinzând WSDL. Implementarea efectivă nu are importanță în acest context.

La finalul acestei secțiuni, furnizăm maniera de descriere semantică a unei instanțe de serviciu Web de furnizare a cantității disponibile dintr-un sortiment de portocale (omitem declarațiile spațiilor de nume și a modului de specificare abstractă a unor resurse):

```
<!-- serviciul Web descris -->
<s:Service ID="FurnizorPortocale">
   <!-- prezintă un profil -->
   <s:presents
      resource="#profil_Furnizor"/>
   <!-- este descris de un proces -->
   <s:describedBy resource="#proces_Furnizor"/>
   <!-- se bazează pe o descriere-suport -->
   <s:supports resource="#desc_Furnizor"/>
</s:Service>
<!-- profilul unui serviciu -->
<p:Profile ID="profil_Furnizor">
   <s:presentedBy resource="#FurnizorPortocale"/>
   <!-- are asociat un proces -->
```

```
    <p:has_process resource="#proces_Furnizor" />
    <!-- posedă un nume -->
    <p:serviceName>FurnizeazaPortocale</p:serviceName>
    <!-- are o intrare -->
    <p:hasInput resource="#intrare" />
    <!-- ...şi o ieşire (rezultat) -->
    <p:hasOutput resource="#rezultat" />
</p:Profile>
<!-- procesul atomic reprezentând serviciul Web -->
<a:AtomicProcess ID="proces_Furnizor">
    <s:describes resource="#FurnizorPortocale" />
    <!-- descrierea datelor de intrare -->
    <a:hasInput>
      <a:Input ID="intrare">
        <!-- de la intrare se aşteaptă
             un şir de caractere -->
        <a:parameterType>xsd:string</a:parameterType>
        <!-- semantic, reprezintă o categorie
             de sortiment -->
        <a:semanticType resource="Sortiment" />
      </a:Input>
  </a:hasInput>
  <!-- descrierea datelor de ieşire -->
  <a:hasOutput>
    <a:Output ID="rezultat">
      <!-- rezultatul întors va fi un întreg -->
      <a:parameterType>xsd:integer</a:parameterType>
      <!-- conceptual, reprezintă o cantitate -->
      <a:semanticType resource="Cantitate" />
    </a:Output>
  </a:hasOutput>
</a:AtomicProcess>
```

După cum se poate remarca, principalele avantaje aduse de aceste specificații conceptuale sunt cele privind automatizarea descoperirii, selectării, invocării, compunerii și monitorizării execuției serviciilor Web.

Referințe

Abbas, A., *Grid Computing: A Practical Guide to Technology and Applications*, Charles River Media, 2004

Acostăchioaie, D., *Securitatea sistemelor Linux*, Polirom, Iași, 2003

Albin, S., *The Art of Software Architecture: Design Methods and Techniques*, Wiley & Sons, 2003

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006: http://www.infoiasi.ro/~busaco/books/xml/

Buraga, S., *Proiectarea siturilor Web* (ediția a doua), Polirom, Iași, 2005: http://www.infoiasi.ro/~design/

Buraga, S., *Semantic Web*, Matrix Rom, București, 2004: http://www.infoiasi.ro/~sweb/

Buraga, S. (coord.), *Tendințe actuale în proiectarea și dezvoltarea aplicațiilor Web*, Matrix Rom, București, 2006: http://www.infoiasi.ro/~busaco/books/nw05/

Cabrera, L.F., Kurt, C., *Web Services Architecture and Its Specifications*, Microsoft Press, 2005

Coyle, F., *XML, Web Services, and the Data Revolution*, Addison-Wesley, 2002

Dierks, T., Allen., C., *The TLS Protocol Version 1.0*, RFC 2246, Internet Engineering Task Force (IETF), 1999: http://www.ietf.org/rfc/rfc2246.txt

Eastlake, D., Reagle, J. (eds.), *XML Encryption Syntax and Processing*, W3C Recommendation, Boston, 2002: http://www.w3.org/TR/xmlenc-core/

Eastlake, D., Reagle, J., Solo, D. (eds.), *XML-Signature Syntax and Processing*, W3C Recommendation, Boston, 2002: http://www.w3.org/TR/xmldsig-core/

Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, 2005

Fisher, M. *et al.*, *Java EE and .NET Interoperability*, Prentice Hall, 2006

Guruge, A., *Web Services: Theory and Practice*, Digital Press, 2004

Hallam-Baker, P., Mysore, S. (eds.), *XML Key Management Specification (XKMS 2.0)*, W3C Recommendation, Boston, 2005: http://www.w3.org/TR/xkms2/

Hartman, B. *et al.*, *Mastering Web Services Security*, Wiley Publishing, 2003

Hundhausen, R., *Working with Microsoft Visual Studio 2005 Team System*, Microsoft Press, 2006

Joseph, K., Fellenstein, C., *Grid Computing*, Prentice Hall, 2003

Kavantzas, N. *et al.* (eds.), *Web Services Choreography Description Language (WS-CDL) Version 1.0*, W3C Candidate Recommendation, Boston, 2005: http://www.w3.org/TR/ws-cdl-10/

Mahmoud, Q., *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)*, Sun, 2005: http://java.sun.com/developer/technical/WebServices/soa/

McLaren, C. (ed.), *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML)*, OASIS Standard, 2002: http://www.oasis-open.org/committees/security/docs/

O'Neill, M. *et al.*, *Web Services Security*, McGraw-Hill/Osborne, 2003

Von Laszewski, G., Wagstrom, P., „Gestalt of the Grid", în Hariri, S., Parashar, M. (eds.), *Tools and Environments for Parallel and Distributed Computing*, Wiley Publishing, 2004

Weerawarana, S. *et al.*, *Web Services Platform Architecture*, Prentice Hall PTR, 2005

* * *, *Apache Tuscany*: http://incubator.apache.org/tuscany/

* * *, *Business Process Execution Language for Web Services*: http://www.oasis-open.org/committees/tc-home.php/wg_abbrev=wspel

* * *, *Eclipse SOA Tools Project*: http://www.eclipse.org/stp/

* * *, *Global Grid Forum*: http://www.globalgridforum.org/

* * *, *Globus*: http://www.globus.org/
* * *, *IBM's DeveloperWorks Web Services*: http://www-136.ibm.com/developerworks/webservices
* * *, *Inițiativa Electronic Business XML*: http://www.ebxml.org/
* * *, *Inițiativa WS-I*: http://www.ws-i.org/
* * *, *Microsoft WSE (Web Services Enhancements)*: http://msdn.microsoft.com/webservices/building/wse/
* * *, *OWL-S*: http://www.daml.org/services/owl-s/1.1/overview/
* * *, *Proiectul GlassFish*: http://java.sun.com/javaee/GlassFish
* * *, *Proiectul WSRF.NET*: http://www.cs.virginia.edu/~gsw2c/WSRFdotNet/
* * *, *Security in a Web Services World: A Proposed Architecture and Roadmap*, IBM & Microsoft, 2002: http://www-106.ibm.com/developerworks/library/ws-secmap/
* * *, *Situl dedicat serviciilor Web*: http://www.webservices.org/
* * *, *Soluțiile de securitate XML oferite de Verisign*: http://www.verisign.com/xml/
* * *, *SWSF (Semantic Web Services Framework)*: http://www.daml.org/services/swsf/1.0/overview/
* * *, *Web Services Activity*: http://www.w3.org/2002/ws/
* * *, *Web Services Addressing*: http://www.w3.org/Submission/ws-addressing/
* * *, *Web Services AtomicTransactions*: http://www.ibm.com/developerworks/webservices/library/ws-atomtran
* * *, *Web Services Business Activity*: http://www.ibm.com/developerworks/webservices/library/ws-busact
* * *, *Web Services Coordination*: http://www.ibm.com/developerworks/webservices/library/ws-coor
* * *, *Web Services Dynamic Discovery*: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-discovery.pdf
* * *, *Web Service Enumeration*: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-enumeration.pdf
* * *, *Web Services Eventing*: http://xml.coverpages.org/WS-Eventing200408.pdf
* * *, *Web Services Interoperability Technology*: http://wsit.dev.java.net/
* * *, *Web Services Metadata Exchange*: ftp://www6.software.ibm.com/software/developer/library/WS-MetadataExchange.pdf
* * *, *Web Service Modeling Ontology*: http://www.wsmo.org/
* * *, *Web Services Policy Assertions Language*: http://www-106.ibm.com/developerworks/library/ws-polas
* * *, *Web Services Policy Attachment*: http://www-106.ibm.com/developerworks/webservices/library/ws-polatt
* * *, *Web Services Resource Framework*: http://www.globus.org/wsrf/
* * *, *Web Services Security: SOAP Message Security*: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
* * *, *Web Service Transfer*: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-transfer.pdf
* * *, *WFMC (WorkFlow Management Coalition)*: http://www.wfmc.org/
* * *, *World Wide Web Consortium*, Boston, 2006: http://www.w3.org/

Acronime

A2A – Application to Application
ACID – Atomicity, Consistency, Isolation, Durability
ACL – Access Control List
AHAH – Asynchronous HTML and HTTP
AJAX – Asynchronous JavaScript and XML
ALE – AJAX Linking and Embedding
ANSI – American National Standards Institute
AMI – Application Messaging Interface
API – Application Programming Interface
ASAP – Asynchronous Service Access Protocol
ASCII – American Standard Code for Information Interchange
ASF – Apache Software Foundation
ASP – Active Server Pages/Application Service Provider
AT – Atomic Transaction
AWS – Amazon Web Services
B2B – Business to Business
B2Bi – Business to Business Integration
BASH – Bourne Again SHell
BSD – Berkeley Software/Standard Distribution
BPEL – Business Process Execution Language
BPEL4WS – Business Process Execution Language for Web Services
BPM – Business Process Management
BTP – Business Transaction Protocol
CA – Certification Authority
CC – Creative Commons
CC/PP – Composite Capability/Preference Profiles
CGI – Common Gateway Interface
CLR – Common Language Run-time
CMS – Content Management System
CoG – Commodity Grid
COM – Common Object Model
CORBA – Common Object Request Broker Architecture

CPAN – Comprehensive Perl Archive Network
CRUD – Create, Retrieve, Update, Delete
C/S – Client/Server
CSS – Cascading Style Sheets
CVS – Control Versioning System
DAIS – Data Access and Integration Services
DAO – Data Access Object
DBMS – DataBase Management System
DCOM – Distributed COM
DDoS – Distributed Denial of Service
DHTML – Dynamic HTML
DIME – Direct Internet Message Encapsulation
DOM – Document Object Model
DoS – Denial of Service
DRM – Distributed Resource Management, Digital Rights Management
DSML – Directory Services Markup Language
DSSO – Desktop Single Sign-On
DTD – Document Type Definition
DWR – Direct Web Remoting
E4X – ECMAScript for XML
EAI – Enterprise Application Integration
EASI – Enterprise Application Security Integration
ebXML – Electronic Business XML
ECMA – European Computer Manufacturers Association
EIS – Enterprise Information System
EJB – Enterprise JavaBean
ESB – Enterprise Service Bus
FAT – File Allocation Table, Fade Anything Technique
FTP – File Transfer Protocol
GCC – GNU Compiler Collection
GGF – Global Grid Forum
GNU – Gnu is Not Unix
GPL – GNU General Public License
GSH – Grid Service Handler
GUID – Globally Unique IDentifier
GWT – Google Web Toolkit
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
i8n – Internationalization
IDE – Integrated Development Environment
IDL – Interface Definition Language
IDN – International Domain Names
IE – Internet Explorer

IETF – Internet Engineering Task Force
IIS – Internet Information Services
IL – Intermediate Language
IRI – Internationalized Resource Identifier
IP – Internet Protocol
IPSec – Internet Protocol Security
IPv6 – Internet Protocol version 6
ISO – International Standards Organization
J2EE – Java 2 Platform, Enterprise Edition
J2SE – Java 2 Platform, Micro Edition
J2SE – Java 2 Platform, Standard Edition
JAAS – Java Authentication and Authorization Service
JAX – Java APIs for XML
JAXB – Java Architecture for XML Binding
JAXP – Java API for XML Processing
JAX-RPC – Java API for XML-based RPC
JAX-WS – Java APIs for XML Web Services
JBI – Java Business Integration
JCP – Java Community Process
JDBC – Java DataBase Connectivity
JDK – Java Development Kit
JIT – Just-In-Time
JMS – Java Message Service
JRE – Java Run-time Environment
JS – JavaScript
JSF – JavaServer Faces
JSON – JavaScript Object Notation
JSP – JavaServer Pages
JSR – Java Specification Repository/Request
JSRS – JavaScript Remote Scripting
JVM – Java Virtual Machine
JWSDP – Web Services Developer Pack
LAMP – Linux-Apache-MySQL-Perl/PHP/Python
MDI – Multiple Document Interface
MEP – Message Exchange Pattern
MIME – Multipurpose Internet Mail Extensions
MQ – Message Queue
MSDN – MicroSoft Developer Network
MTOM - Message Transmission Optimization Mechanism
MVC – Model-View-Controller
MXML – Maximum Experience Markup Language
NAICS – North American Industry Classification System
OASIS – Organization of the Advancement of Structured Information Standards

ODBC – Open DataBase Connectivity
OMG – Object Management Group
OOP – Object Oriented Programming/Paradigm
ORM – Object-Relational Mapping
OS – Operating System
OWL – Web Ontology Language
P2P – Peer-to-Peer
PDF – Portable Document Format
PEAR – PHP Extensions and Add-ons Repository
PERL – Practical Extraction and Report Language
PHP – PHP: Hypertext Preprocessor
POD – Process-On-Demand
POJO – Plain Old Java Object
POX – Plain Old XML
PSVI – Post-Schema Validation Infoset
RA – Registration Authority
RAI – RSS Abstraction Interface
RBAC – Role-Based Access Control
RDF – Resource Description Format
RDFS – RDF Schema
REST – REpresentational State Transfer
RoR – Ruby on Rails
QoS – Quality of Service
RAD – Rapid Application Development
RFC – Request For Comments
RIA – Rich Internet Application
RMI – Remote Method Invocation
RPC – Remote Procedure Call
RSS – Rich/RDF Site Summary, Really Simple Syndication
SAAJ – SOAP with Attachments API for Java
SAML – Security Assertion Markup Language
SAX – Simple API for XML
SCA – Service Component Architecture
SDI – Single Document Interface
SDK – Software Development Kit
SDL – Service Development Language
SDO – Service Data Objects
SEI – Service Endpoint Interface
SIG – Special Interest Group
SLA – Service Level Agreement
SMTP – Simple Mail Transfer Protocol
SOA – Service-Oriented Architecture
SOAD – Service-Oriented Analysis and Design

SOAP – Simple Object Access Protocol
SOM – Service-Oriented Model
SOMA – Service-Oriented Modelling and Architecture
SPI – Service Provider Interface
SQL – Structured Query Language
SRMP – SOAP Reliable Messaging Protocol
SSI – Server-Side Includes
SSL – Secure Sockets Layer
SSO – Single Sign-On
SW – Semantic Web
SwA – SOAP with Attachments
SWS – Semantic Web Service
SWSF – Semantic Web Services Framework
TCP – Transmission Control Protocol
TLS – Transport Layer Security
TR – Technical Report
UA – User Agent (Web)
UBL – Universal Business Language
UBR – UDDI Business Registry
UDDI – Universal Description, Discovery, and Integration
UDP – User Datagram Protocol
UI – User Interface
UML – Unified Modeling Language
UPnP – Universal Plug and Play
URI – Uniform Resource Identifier
URL – Uniform Resource Locator
URN – Uniform Resource Name
UTF – Unicode Transformation Format
UUID – Universal Unique IDentifier
VB – Visual Basic
VM – Virtual Machine
VPN – Virtual Private Network
W3C – World Wide Web Consortium
WAP – Wireless Application Protocol
WCF – Windows Communications Foundation
WDDX – Web Distributed Data Exchange
WML – Wireless Markup Language
WS – Web Service
WSA – Web Service Architecture/Addressing
WSC – Web Service Choreography
WSCI – Web Service Choreography Interface
WSDL – Web Services Description Language
WSDM – Web Service Distributed Management

WS-DP – Device Profile for Web Services
WSE – Web Services Enhancements
WSF – Web Services Framework
WSFL – Web Services Flow Language
WS-I – Web Services Interoperability
WSIA – Web Services for Interactive Applications
WSIL – Web Service Inspection Language
WSIT – Web Services Interoperability Technology
WSMO – Web Service Modeling Ontology
WSO – Web Services Orchestration
WS-PA – WS-Policy Attachment
WSRF – Web Services Resource Framework
WSRP – Web Services for Remote Portals
WSS – Web Service Security
WSTK – Web Services Toolkit
WSXL – Web Services Experience Language
WWW – World Wide Web
XAML – Extensible Application Markup Language
XACML – Extensible Access Control Markup Language
XHTML – Extensible HyperText Markup Language
X-KISS – XML Key Information Service Specification
XKMS – XML Key Management Specification
X-KRSS – XML Key Registration Service Specification
XML – Extensible Markup Language
XMLNS – XML NameSpace
XMLP – Extensible Markup Language Protocol
XP – Extreme Programming
XSD – XML Schema Definition/Datatypes
XSL – Extensible Stylesheet Language
XSLT – XSL Transformation
XSS – Cross-Site Scripting

Bibliografie generală

Abbas, A., *Grid Computing: A Practical Guide to Technology and Applications*, Charles River Media, 2004

Anghel, T., *Programarea în PHP. Ghid practic*, Polirom, Iași, 2005

Asleson, R., Schutta, N., *Foundations of AJAX*, Apress, 2006

Buraga, S., *Tehnologii XML*, Polirom, Iași, 2006: http://www.infoiasi.ro/~busaco/books/xml/

Buraga, S.,*Proiectarea siturilor Web* (ediția a doua), Polirom, Iași, 2005: http://www.infoiasi.ro/~design/

Buraga, S., *Semantic Web*, Matrix Rom, București, 2004: http://www.infoiasi.ro/~sweb/

Buraga, S. (coord.), *Tendințe actuale în proiectarea și dezvoltarea aplicațiilor Web*, Matrix Rom, București, 2006: http://www.infoiasi.ro/~busaco/books/nw05/

Buraga, S. (coord.), *Situri Web la cheie. Soluții profesionale de implementare*, Polirom, Iași, 2004: http://www.infoiasi.ro/~busaco/books/webapps/

Cabrera, L.F., Kurt, C., *Web Services Architecture and Its Specifications*, Microsoft Press, 2005

Daconta, M., Obrst, L., Smith, K., *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, John Wiley & Sons, 2003

Daum, B., Merten, U., *System Architecture with XML*, Elsevier Science, 2003

Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, 2005

Geroimenko, V., *Dictionary of XML Technologies and the Semantic Web*, Springer-Verlag, 2004

Guruge, A., *Web Services: Theory and Practice*, Digital Press, 2004

Hartman, B. *et al.*, *Mastering Web Services Security*, Wiley Publishing, 2003

Harold, E., *XML 1.1 Bible* (3rd Edition), Wiley Publishing, 2004

Kulchenko, P., Ray, R., *Programming Web Services with Perl*, O'Reilly, 2002

Linthicum, D., *Enterprise Application Integration*, Addison-Wesley, 1999

Myerson, J., *The Complete Book of Middleware*, Auerbach Publications, 2002

Mueller, J., *Mining Google Web Services: Building Applications with the Google API*, Sybex, 2004

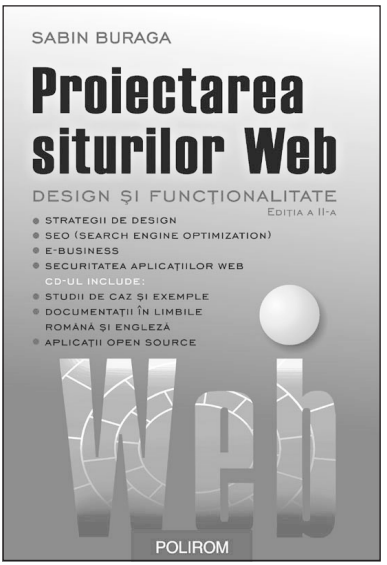
Newcomer, E., Lomow, G., *Understanding SOA with Web Services*, Addison-Wesley, 2004

O'Neill, M. *et al.*, *Web Services Security*, McGraw-Hill/Osborne, 2003

Perry, B., *AJAX Hacks*, O'Reilly, 2006

Powell, T., Schneider, F., *JavaScript 2.0: The Complete Reference* (Second Edition), McGraw-Hill/Osborne, 2004
Robinson, S. *et al.*, *Professional C#* (Third Edition), Wiley Publishing, 2004
Shiflett, C., *HTTP Developer's Handbook*, Sams Press, 2003
Si Shi, N., Murthy, V.K., *Architectural Issues of Web-Enabled Electronic Business*, Idea Group Publishing, 2003
Sklar, D., Trachtenberg, A., *PHP Cookbook*, O'Reilly, 2002
Tanasă, Ş., Olaru, C., Andrei, Ş., *Java de la 0 la expert*, Polirom, Iaşi, 2003
Tidwell, D., Snell, J., Kulchenko, P., *Programming Web Services with SOAP*, O'Reilly, 2001
Topley, K., *Java Web Services in Nutshell*, O'Reilly, 2003
Waltz, E., *Knowledge Management in the Intelligence Enterprise*, Artech House, 2003
Weerawarana, S. *et al.*, *Web Services Platform Architecture*, Prentice Hall PTR, 2005
Zakas, N., McPeak, J., Fawcett, J., *Professional AJAX*, Wrox Press, 2006
Zeldman, J., *Designing with Web Standards*, New Riders Press, 2003
* * *, *Apache XML*: http://www.apache.com/
* * *, *API-urile puse la dispoziţie de Google*: http://www.google.com/apis/
* * *, *Apple Developer Connection*: http://developer.apple.com/
* * *, *ASP.NET*: http://www.asp.net/
* * *, *Café con Lèche*: http://www.cafeconleche.org/
* * *, *Code Project*: http://www.codeproject.com/
* * *, *Developers.net*: http://www.developers.net/
* * *, *Dezvoltarea de aplicaţii SOAP pentru Pocket PC*: http://www.pocketsoap.com/
* * *, *ebXML*: http://www.ebxml.org/
* * *, *Fundaţia Mozilla*: http://www.mozilla.org/
* * *, *IBM Developer's Site*: http://www.ibm.com/developer/
* * *, *Iniţiativa WS-I*: http://www.ws-i.org/
* * *, *Instrumente XML disponibile la Apache*: http://xml.apache.org/
* * *, *Interop Month*: http://www.interopmonth.com/
* * *, *Java Coffee-Break*: http://www.javacoffeebreak.com/
* * *, *Java & Web Services*: http://java.sun.com/webservices/
* * *, *JCP (Java Community Process)*: http://jcp.org/
* * *, *Lista specificaţiilor privitoare la limbajele bazate pe XML*: http://xml.coverpages.org/
* * *, *Microsoft Developers Network*: http://msdn.microsoft.com/
* * *, *Mozilla Developer Center*: http://developer.mozilla.org/
* * *, *O'Reilly OnLAMP*: http://www.onlamp.com/
* * *, *O'Reilly OnDotNET*: http://www.ondotnet.com/
* * *, *O'Reilly OnJava*: http://www.onjava.com/
* * *, *Organizaţia OASIS (Organization for the Advancement of Structured Information Standards)*: http://www.oasis-open.org/
* * *, *OSI (Open Source Initiative)*: http://www.opensource.org/
* * *, *Planet Source Code*: http://www.planet-source-code.com/
* * *, *Perfect XML*: http://www.perfectxml.com/
* * *, *Perl*: http://www.perl.com/

* * *, *Perl XML*: http://www.perlxml.com/
* * *, *PHP Builder*: http://www.phpbuilder.com/
* * *, *PHP Classes*: http://www.classes.net/
* * *, *PHP Developer*: http://www.phpdeveloper.org/
* * *, *Portal dedicat serviciilor Web*: http://www.webservices.org/
* * *, *Rapoartele tehnice ale Consorțiului Web*: http://www.w3.org/TR/
* * *, *Resurse privind programare pe partea de server*: http://www.devshed.com/Server_Side/
* * *, *Situl .NET 247*: http://www.dotnet247.com/
* * *, *Situl dedicat dezvoltatorilor Java*: http://developer.java.sun.com/
* * *, *Situl dezvoltatorilor .NET*: http://www.gotdotnet.com/
* * *, *Situl lui Paul Prescod*: http://www.prescod.net/
* * *, *Situl privitor la șabloanele de proiectare a aplicațiilor AJAX*: http://www.ajaxpatterns.org/
* * *, *Situl proiectului Mono*: http://www.mono-project.com/
* * *, *SOA Web Services Journal*: http://webservices.sys-con.com/
* * *, *SOAP Software*: http://www.soapware.org/
* * *, *Source Forge*: http://www.sourceforge.net/
* * *, *Sun XML Developer Connection*: http://www.sun.com/software/xml/developers/
* * *, *The Open Sourcery*: http://theopensourcery.com/
* * *, *WebReference*: http://www.webreference.com/
* * *, *Wikipedia*: http://www.wikipedia.org/
* * *, *XFront*: http://www.xfront.com/
* * *, *XMethods*: http://www.xmethods.net/
* * *, *XML 101*: http://www.xml101.com/
* * *, *XML.com*: http://xml.com/
* * *, *XML.org*: http://www.xml.org/
* * *, *XML Software*: http://www.xmlsoftware.com/
* * *, *XML Standards Library*: http://xmlstds.xemantics.com/
* * *, *ZVON*: http://www.zvon.org/



Sabin Buraga

Proiectarea siturilor Web. Design și funcționalitate
(cartea include CD) (ediția a II-a)

Lucrarea realizează mai mult decât o introducere în domeniul Web, ilustrând potențialul acestuia de a dezvolta aplicații puternice din domenii cum ar fi informarea și documentarea, realitatea virtuală, *e-business*. Cititorul va putea înțelege și exploata posibilitățile oferite de noile tehnologii în ceea ce privește găsirea de informații, demararea de afaceri *online* și proiectarea eficientă a siturilor Web.

Principalele noutăți aduse de ediția a II-a: Strategii de optimizare a siturilor (*SEO – Search Engine Optimization*) • Utilizabilitatea, ergonomia și accesibilitatea siturilor Web • Securitatea în spațiul Web • Forumuri, *blog*-uri, situri *wiki* și portaluri • *E-branding, e-government, e-learning* • Prin XML, spre Web-ul semantic

Conținutul CD-ului: Studii de caz și exemple • Documentații (tutoriale, manuale, specificații) în limbile română și engleză • Curs de tehnologii Web – 14 prezentări și demonstrații (în limba română) • Aplicații *open source* pentru comunități virtuale și comerț electronic

www.polirom.ro

Coperta: Radu Răileanu
Tehnoredactor: Gabriela Gheţău

Bun de tipar: octombrie 2006. Apărut: 2006
Editura Polirom, B-dul Carol I nr. 4 • P.O. Box 266
700506, Iași, Tel. & Fax (0232) 21.41.00; (0232) 21.41.11;
(0232) 21.74.40 (difuzare); E-mail: office@polirom.ro
București, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33,
O.P. 37 • P.O. Box 1-728, 030174
Tel.: (021) 313.89.78; E-mail: office.bucuresti@polirom.ro

Tiparul executat la S.C. LUMINA TIPO s.r.l.
str. Luigi Galvani nr. 20 bis, sect. 2, București
Tel./Fax: 211.32.60, 212.29.27, E-mail: office@luminatipo.com